VIVO 1.14.x Documentation

VIVO 1.14.x Documentation

Exported on 05/24/2024

Table of Contents

1 Introduction 26
1.1 What is VIVO? 26
1.2 Release Notes 26
1.2.1 Overview 26
1.2.2 What's New 27
1.2.2.1 Simplified process for installation and customization 27
1.2.2.2 Eliminating language specific FTL files 27
1.2.2.3 Simplified process of supporting new languages 27
1.2.2.4 Improvement of performance for startup of VIVO 27
1.2.2.5 Faux data properties 27
1.2.2.6 Configuration of usage of Map of Science visualization 27
1.2.3 Resolved Issues 28
1.2.4 Bug 28
1.2.5 New Feature 28
1.2.6 Improvement 29
1.2.7 Release managers 30
1.2.8 Acknowledgements 30
1.3 Functional Overview 30
1.3.1 Online Access 30
1.3.1.1 Linked Open Data 30
1.3.1.2 Built-in Search 31
1.3.1.3 Navigation 31
1.3.1.4 Optimisations for Google Indexing 31
1.3.1.5 Support for Modern Browsers 31
1.3.2 Getting Data into VIVO 31
1.3.2.1 Manual Data Entry 31
1.3.2.2 Automated Data Entry 31

1.3.3 Access Control 31
1.4 System Requirements 32
1.4.1 Hardware Recommendations 32
1.4.1.1 Minimum Specification 32
1.4.1.2 Recommended Specification 32
1.4.2 Software Requirements 32
1.4.2.1 Operating System 32
1.4.2.2 Java 8 or 11 33
1.4.2.3 Maven 3.6.1 or later 33
1.4.2.4 Tomcat 8 or 9 34
1.4.2.5 Optional - MySQL / MariaDB 5.5 or later (or any other supported by Jena SDB) 34
1.4.3 Running VIVO behind an Apache server 35
2 Installing VIVO 37
2.1 Installing from Distribution 37
2.1.1 Overview 37
2.1.2 Preparing the Installation Settings 37
2.1.3 Permissions 38
2.1.4 Installing VIVO 38
2.2 Installing from GitHub 39
2.2.1 Preparing the Repositories 39
2.2.2 Preparing the Installation Settings 39
2.2.3 Permissions 39
2.2.4 Installing VIVO 40
2.2.4.1 Default Installer 40
2.2.4.2 Custom Installer 40
2.3 Completing the Installation 41
2.3.1 Configure the Home Directory 41
2.3.2 Configure the Database Schema 41

2.3.3 Configure and Start Solr ... 41
2.3.4 Configure and Start Tomcat ... 42
2.3.4.1 Set JVM parameters ... 42
2.3.4.2 Set security limits ... 43
2.3.4.3 Set URI encoding ... 43
2.3.4.4 Take care when creating Context elements ... 44
2.3.4.5 Starting Tomcat ... 44
2.4 Verify Your Installation ... 44
3 Upgrading VIVO ... 46
3.1 Upgrading from VIVO 1.10.x prior versions ... 46
3.2 Additional Considerations ... 46
3.2.1 Overview ... 46
3.2.2 Upgrade Local Java Code Using Jena ... 47
3.2.3 Notes on Other Dependency Changes ... 47
3.2.4 UI Changes ... 47
3.2.4.1 jQuery 1.12.4 ... 47
3.2.4.2 jQuery plugins ... 48
3.2.4.3 D3 v4 ... 48
3.2.5 List View Configurations ... 49
3.2.6 Servlet 3.0 Upgrade ... 50
3.2.7 Java Dependencies ... 50
3.2.7.1 HttpClient ... 50
3.2.7.2 OSGi Dependencies ... 50
3.2.7.3 JSON Parsers ... 51
3.2.7.4 Replaced Dependencies ... 51
3.2.7.5 Removed Dependencies ... 51
3.3 Preserving Customizations During Build ... 51
3.3.1 Overview ... 51
3.3.2 Maven Custom Installer ... 52

3.3.3 Example – Custom Theme 52
3.3.4 Example – Local Ontology 52
3.3.5 Example – Data Distribution API 52
4 Exploring VIVO 53
4.1 Overview 53
4.2 Logging in to VIVO 53
4.3 Sample Data 54
4.3.1 Overview 54
4.3.2 Preparing Your VIVO 54
4.3.3 Loading the Sample Data 55
4.3.4 Exploring the Interface 57
4.3.5 Resetting Your Database 61
4.4 Restoring VIVO to First Time State 61
5 Preparing for Production 63
5.1 Overview 63
5.2 Minimum Configuration 63
5.2.1 Email 63
5.2.2 Namespace 64
5.2.3 Additional Configuration 64
5.3 About Page and Support Link 64
5.3.1 Overview 64
5.3.2 About Page 65
5.3.3 Support Link 65
5.4 Create, Assign, and Use an Institutional Internal Class 65
5.4.1 Overview 65
5.4.2 Create an Institutional Internal Class 66
5.4.3 Assign your Institutional Internal Class 67
5.4.4 Use your Institutional Internal Class 68
5.5 Adding User Accounts 68

5.6 Adding Site Information ... 69
5.7 Simple Theming ... 70
5.7.1 Overview ... 70
5.7.2 Create a Theme ... 70
5.7.3 Modify Header with Logo ... 71
5.7.4 Modify Footer ... 71
5.7.5 Add Your Colors ... 72
5.7.6 Additional Theming ... 72
5.7.7 Experimental Theming ... 72
5.8 Configuring Web Analytics Tools ... 73
5.8.1 Background ... 73
5.8.2 Example: Google Analytics ... 73
5.8.2.1 googleAnalytics.ftl ... 73
5.8.3 Other analytics platforms ... 74
5.9 Adding the Data Distribution API ... 74
5.9.1 Overview ... 74
5.9.2 Process for Adding the Data Distribution API ... 74
5.10 Configuring External Vocabularies ... 74
5.10.1 Overview ... 74
5.10.2 Configuring the UMLS External Vocabulary Service ... 75
5.10.3 Notes ... 75
6 Using VIVO ... 76
6.1 Navigating VIVO ... 76
6.2 Editing Your Profile ... 77
6.2.1 Overview ... 78
6.2.2 Sign on to VIVO ... 78
6.2.3 Adding Your Photo ... 80
6.2.4 Adding an Overview ... 82
6.2.5 Positions and Publications ... 83

6.2.5.1 Harvesting publication metadata from Crossref and PubMed 83
6.2.6 Additional Items for Your Profile 84
6.3 Using Search 84
6.3.1 Searching 85
6.3.2 Search results 86
6.4 Using Visualizations 87
6.4.1 Visualizations in VIVO 87
6.4.2 Co-Authorship and Co-Investigator Network Visualizations 87
6.4.3 Map of Science 88
6.4.4 Using the Capability Map 89
6.4.4.1 Overview 89
6.4.4.2 A Tour of the Capability Map 89
6.4.5 Using the Temporal Graph 94
6.5 VIVO for Data Analysts 95
6.5.1 Background 95
6.5.2 Getting Rectangles of Data 96
6.5.3 Getting Graphs of Data 96
6.5.4 Repeatable Processes 96
6.5.5 Distributed Queries 96
6.5.6 References 96
7 Managing Data in Your VIVO 97
7.1 Importing Data to VIVO 97
7.1.1 Using the Convert CSV to RDF ingest tool 97
7.1.1.1 Mapping Ontologies to other Ontologies 98
7.1.1.2 Example workflow 98
7.1.1.3 Appendix A: SPARQL Queries 105
7.1.2 Data types for string and language 107
7.1.2.1 Literal Values 107
7.1.2.2 Recommendations 108

7.2 Exporting Data from VIVO 109
7.2.1 Exporting All Data 109
7.2.2 Exporting Selected Data 109
7.3 Managing Person Identifiers 109
7.3.1 Notes 110
7.4 Managing Organization Hierarchy 111
7.4.1 Overview 111
7.4.2 "hasPart, partOf" 111
7.4.3 Your Organizational Data 111
7.4.4 Making Triples 112
7.4.4.1 Notes regarding the triples 114
7.4.5 Managing the Triples in VIVO 114
7.4.5.1 Add the triples to VIVO 114
7.4.5.2 Updating your triples in VIVO 116
7.4.6 Some Closing Observations 116
7.5 Managing Data Packages 116
7.5.1 Overview 116
7.5.2 Add a data package 117
7.5.3 Update a data package 117
7.5.4 Delete a data package 117
7.5.5 Available Data Packages 118
7.6 SPARQL Queries 118
7.6.1 Overview 118
7.6.2 Running SPARQL queries 118
7.6.3 Using SPARQL for reporting 120
7.6.4 Using SPARQL to clean data 121
7.6.5 DESCRIBE queries 121
7.6.6 ASK Queries 122
7.6.7 Additional SPARQL Resources 123

7.7 How to remove data from a specific graph ... 123
7.8 Removing Entities from VIVO ... 123
7.8.1 General Method ... 123
7.8.2 Examples ... 124
7.8.2.1 Remove publications by type ... 124
7.8.2.2 Remove Other Entities ... 126
7.9 Uploading files associated with individuals ... 126
7.9.1 Configuration of the runtime.properties ... 126
7.9.2 Adding support for assigning file to the certain individuals class ... 126
8 Extending and Localizing VIVO ... 128
8.1 Overview ... 128
8.2 Internationalization ... 129
8.2.1 Children Pages ... 129
8.2.2 Summary of this Page ... 129
8.2.3 VIVO Language Support ... 129
8.2.4 Adding an existing language to your VIVO site ... 130
8.2.5 Building VIVO and Vitro language repositories from source (for developers) ... 130
8.2.6 Creating new language files for your language ... 130
8.2.6.1 The locale ... 131
8.2.6.2 The language files ... 131
8.2.7 How VIVO supports languages ... 133
8.2.7.1 Language in the data model ... 133
8.2.7.2 Language support in VIVO pages ... 134
8.2.7.3 Language in Freemarker page templates ... 138
8.2.7.4 Language-specific templates ... 139
8.2.7.5 Language in Java code ... 139
8.2.7.6 Language in JSPs ... 139
8.2.7.7 Language in JavaScript files ... 139

8.2.8 Enabling Interface Languages in VIVO as an Administrator 140
8.2.8.1 Quick Start 140
8.2.8.2 VIVO i18n Design 141
8.2.9 Using VIVO's Internationalization (i18n) Features 141
8.2.9.1 Summary of language support... 141
8.2.9.2 Selecting a language 142
8.2.9.3 Adding content in multiple languages 142
8.2.9.4 Editing content in multiple languages 147
8.2.9.5 Troubleshooting language related problems 148
8.2.10 Vitro UI labels vocabulary 148
8.3 Customizing the Interface 150
8.3.1 Introduction 150
8.3.1.1 Making changes to VIVO 150
8.3.1.2 VIVO is already customized 150
8.3.2 Adding your own customizations 151
8.3.2.1 Working in the GUI 151
8.3.2.2 RDF files.. 151
8.3.2.3 Changes to the source files... 151
8.3.3 Tool summary 151
8.3.3.1 Required skills 151
8.3.3.2 The tools.. 152
8.3.4 Home page customizations 154
8.3.4.1 Introduction 154
8.3.4.2 The page-home.ftl Template File 154
8.3.4.3 The Research Section... 155
8.3.4.4 The Faculty Section 156
8.3.4.5 The Departments Section.. 156
8.3.4.6 The Geographic Focus Map 157
8.3.5 Menu and page management 163

8.3.5.1 Overview ... 163
8.3.5.2 What to do ... 164
8.3.6 Annotations on the ontology ... 166
8.3.6.1 Edit property groups ... 166
8.3.6.2 Edit the appearance of properties ... 169
8.3.6.3 Create and edit faux properties ... 172
8.3.6.4 Edit class groups ... 177
8.3.6.5 Edit the appearance of classes ... 180
8.3.7 Class-specific templates for profile pages ... 183
8.3.7.1 Overview ... 183
8.3.7.2 How to do it ... 185
8.3.8 Excluding Classes from the Search ... 188
8.3.8.1 Overview ... 188
8.3.8.2 Steps to create a new search exclusion ... 188
8.3.8.3 Example on the contents of an RDF file to define exclusions ... 188
8.3.9 Custom List View Configurations ... 188
8.3.9.1 Introduction ... 188
8.3.9.2 List View Configuration Guidelines ... 189
8.3.9.3 List View Example ... 193
8.3.10 Creating short views of individuals ... 196
8.3.10.1 Overview ... 196
8.3.10.2 Details ... 197
8.3.10.3 Some examples ... 199
8.3.10.4 Troubleshooting ... 208
8.3.10.5 Notes ... 209
8.3.11 Creating a custom theme ... 210
8.3.11.1 Overview ... 210
8.3.11.2 The structure of pages in VIVO ... 213
8.3.11.3 Some significant templates ... 213

8.3.11.4 Making changes 215
8.3.12 Creating custom entry forms 217
8.3.12.1 Overview 217
8.3.12.2 An example 217
8.3.12.3 How is it created? 218
8.3.12.4 Accessing VIVO Data Models 219
8.3.12.5 Implementing custom forms using N3 editing 228
8.3.12.6 Servlet Lifecycle Management 229
8.3.13 Enhancing Freemarker templates with DataGetters 232
8.3.13.1 Overview 232
8.3.13.2 An example 232
8.3.13.3 Creating the DataGetter 233
8.3.13.4 Modifying the template 234
8.3.13.5 Summary 235
8.3.14 Enriching profile pages using SPARQL query DataGetters 235
8.3.14.1 Introduction 235
8.3.14.2 The Steps and an Example 235
8.3.15 Multiple profile types for foaf:Person 240
8.3.15.1 Introduction 240
8.3.15.2 The Profile Page Types 240
8.3.15.3 Implementing Multiple Profile Pages 242
8.3.15.4 Using the Standard View Without Implementing Multiple Profile Pages 245
8.3.16 Using OpenSocial Gadgets 245
8.3.16.1 Overview 245
8.3.16.2 Adding gadgets to VIVO 247
8.3.16.3 Getting started 247
8.3.17 How VIVO creates a page 248
8.3.17.1 The home page 248

8.3.17.2 A profile page ... 250
8.3.17.3 The People page ... 255
8.3.17.4 A back-end page ... 258
8.3.18 Tips for Interface Developers ... 259
8.3.18.1 Use the Developer Panel ... 259
8.3.18.2 Iterate your code more quickly ... 261
8.3.18.3 Reveal what VIVO is doing ... 261
8.4 Adding Additional Ontologies to VIVO ... 262
8.4.1 Overview ... 262
8.4.2 Using the Web Interface ... 263
8.4.2.1 Create an Ontology from the Web Interface ... 263
8.4.2.2 Load an Existing Ontology as Data ... 263
8.4.2.3 Load an Existing Ontology to a Named Graph ... 263
8.4.3 Loading from Filegraphs ... 263
8.4.3.1 Example ... 264
8.4.4 Namespace Prefixes ... 264
8.5 Enable an external authentication system ... 265
8.5.1 How User Accounts are Associated with Profile Pages ... 265
8.5.1.1 A user account may have an externalAuthId ... 265
8.5.1.2 runtime.properties may contain a value for selfEditing.idMatchingProperty ... 266
8.5.1.3 The profile page may match the externalAuthId on the user account ... 266
8.5.2 Shibboleth example ... 266
8.5.2.1 Install the Shibboleth module for Apache ... 267
8.5.2.2 Edit Shibboleth Application Defaults ... 267
8.5.2.3 Secure VIVO's 'special page' at /loginExternalAuthReturn ... 267
8.5.2.4 Allow Shibboleth's pages to be served by Apache ... 267
8.5.2.5 Allow Shibboleth attributes to be passed through to Tomcat ... 268
8.5.2.6 Edit runtime.properties ... 268

8.5.3 Using a Tomcat Realm for external authentication 269
8.5.3.1 Background 269
8.5.3.2 Testing 269
8.6 Authorization 270
8.6.1 Writing a controller for a secured page 270
8.6.1.1 Concepts 270
8.6.1.2 Requested Actions 272
8.6.1.3 The most common case 272
8.6.1.4 A more complex example 274
8.6.2 Creating a VIVO authorization policy - an example 275
8.6.2.1 Overview 275
8.6.2.2 The example 275
8.6.2.3 Setup when VIVO starts 281
8.6.2.4 A more complicated example 283
8.6.3 A more elaborate authorization policy 283
8.6.3.1 The RequestedAction 283
8.6.3.2 The Controller 285
8.6.3.3 The Policy 287
8.6.4 The IdentifierBundle - who is requesting authorization? 289
8.6.4.1 The challenge of identity and authorization 289
8.6.4.2 The IdentifierBundle to the rescue 290
8.7 Adding External Vocabularies 292
8.7.1 Overview 292
8.8 Search Engine Optimization (SEO) 292
8.8.1 Overview 292
8.8.2 Citation Metatags 292
8.8.3 Sitemap 293
8.8.4 Additional SEO Considerations 293
9 System Administration 294

9.1 Background ... 294
9.2 Creating and Managing User Accounts ... 294
9.2.1 Overview ... 294
9.2.2 Authentication ... 295
9.2.2.1 Internal Authentication ... 295
9.2.2.2 External Authentication ... 295
9.2.2.3 External-Only Accounts ... 295
9.2.3 What is a User Account? ... 295
9.2.4 Associating User Accounts with Profile Pages ... 295
9.2.5 User Roles ... 296
9.2.6 The Root User Account ... 296
9.2.7 Managing User Accounts ... 297
9.2.7.1 Normal workflow ... 297
9.2.7.2 Workflow without Email ... 297
9.2.7.3 External Authentication ... 297
9.3 Backup and Restore ... 297
9.4 Inferences and Indexing ... 298
9.4.1 Recompute Inferences ... 298
9.4.2 Re-building the search index ... 298
9.5 The Site Administration Page ... 299
9.5.1 Site Administration ... 299
9.5.2 Data Input ... 299
9.5.3 Ontology Editor ... 300
9.5.3.1 Class Management ... 300
9.5.3.2 Property Management ... 300
9.5.4 Site Configuration ... 301
9.5.4.1 Site Information ... 301
9.5.5 Advanced Tools ... 301
9.5.5.1 Ingest tools ... 302

9.5.6 Site Maintenance 303
9.6 The VIVO log file 303
9.6.1 What does a log message look like? 303
9.6.2 What is the right level for a log message? 304
9.6.3 Setting the output levels 304
9.6.3.1 Production settings 304
9.6.3.2 Developer settings 305
9.6.3.3 Changing levels while VIVO is running 305
9.6.4 Customizing the logging configuration 305
9.6.4.1 Overview 305
9.6.4.2 The default configuration 306
9.6.4.3 Writing some messages to a special log 307
9.6.4.4 More information 308
9.6.5 Writing Exceptions to the Log 308
9.6.5.1 Not the Right Way 308
9.6.5.2 Declaring a Logger 308
9.6.5.3 Bad, Better, Good 309
9.6.5.4 Whoops 310
9.7 Activating the ORCID integration 311
9.7.1 Overview 311
9.7.2 Video tutorial 311
9.7.3 When applying for credentials 312
9.7.3.1 Informing the users 312
9.7.3.2 Connecting to your application 313
9.7.4 Configuring VIVO 313
9.8 Performance Tuning 316
9.8.1 SDB - MySQL Tuning 316
9.8.1.1 Version Recommendation 316
9.8.1.2 MySQL DB Engine 316

9.8.1.3 MySQL Buffers 316
9.8.1.4 Temporary Tables 316
9.8.2 Additional Performance Tips 317
9.8.2.1 What is performance? 317
9.8.2.2 What kind of performance is normal? How do I know if I have a problem? 317
9.8.2.3 Tools for measuring performance 317
9.8.2.4 Tuning for improved performance 318
9.8.3 MySQL tuning, and troubleshooting 320
9.8.3.1 Tuning MySQL 320
9.8.4 Use HTTP caching to improve performance 322
9.8.5 HTTP Cache Awareness (*) 323
9.8.5.1 Overview 323
9.8.5.2 How to enable cache awareness 324
9.8.5.3 What pages can be cached? 324
9.8.5.4 What do the caching headers look like? 324
9.8.5.5 How to configure your cache 324
9.9 Virtual Machine Templates 324
9.9.1 Docker 325
9.9.2 Vagrant 325
9.10 Moving your VIVO Instance 325
9.10.1 Step-by-step guide 325
9.11 Regaining access to the root account 326
9.12 Altmetrics Support 327
9.12.1 Overview 327
9.12.2 Display 327
9.12.3 Configuration 328
9.13 Troubleshooting 328
9.13.1 Having problems with your VIVO installation? 328

9.13.2 Can't find any individuals? 328
9.13.3 Mail not working? 329
9.13.4 Troubleshooting Tips 329
9.13.4.1 Warning screen at startup 329
9.13.4.2 Rebuilding the Search Index 329
9.13.4.3 How to Serve Linked Data 330
9.13.4.4 Long URLS 331
9.14 High Availability 332
9.14.1 Overview 332
9.14.2 Session management 332
9.14.3 Caching 332
9.14.4 Solr 332
9.14.5 Home directory 332
9.14.6 Content triple store 332
9.14.7 Configuration triple store 333
9.15 Replicating Ontology Changes Across Instances 333
9.15.1 Purpose 333
9.15.2 Procedure 333
9.15.3 Best Practice 334
9.16 Jena SDB and MySQL setup 334
9.17 How to mitigate the Log4Shell (CVE-2021-44228, CVSSv3 10.0) vulnerability 336
9.17.1 Log4Shell 336
9.17.2 What is affected 336
9.17.3 Mitigation 336
9.18 How to mitigate the Spring CVE-2022-22965 vulnerability 336
9.18.1 The CVE-2022-22965 vulnerability 336
9.18.2 What is affected 337
9.18.3 Mitigation 337

10 Reference ... 338
10.1 Overview ... 338
10.2 APIs ... 338
10.2.1 Data Distribution API ... 339
10.2.1.1 Overview ... 339
10.2.1.2 Hello, World ... 340
10.2.1.3 Managing Configuration Files ... 341
10.2.1.4 References ... 341
10.2.1.5 Data Distribution Predicates ... 342
10.2.1.6 Data Distributors ... 343
10.2.2 Direct2Experts API ... 345
10.2.2.1 Overview ... 345
10.2.2.2 The Direct2Experts Endpoints ... 345
10.2.2.3 Participating in Direct2Experts ... 345
10.2.2.4 References ... 345
10.2.3 Linked Open Data - requests and responses ... 346
10.2.3.1 Overview ... 346
10.2.3.2 Requesting Linked Open Data from VIVO ... 347
10.2.3.3 What is included in the response? ... 349
10.2.3.4 Restricting properties ... 353
10.2.3.5 Error handling ... 355
10.2.4 ListRDF API ... 355
10.2.4.1 Overview ... 355
10.2.4.2 Specification ... 356
10.2.4.3 Examples ... 358
10.2.5 Reconciliation API ... 359
10.2.5.1 Purpose ... 359
10.2.5.2 Specification ... 359
10.2.5.3 Usage with OpenRefine ... 360

10.2.5.4 Configuration of the VIVO Reconciliation API 362
10.2.6 Search indexing service 362
10.2.6.1 Purpose 362
10.2.6.2 Use Cases 363
10.2.6.3 Indexing and Reasoning 363
10.2.6.4 Specification 363
10.2.6.5 Examples 365
10.2.6.6 Securing the API 365
10.2.7 SPARQL Query API 366
10.2.7.1 Purpose 366
10.2.7.2 Use Cases 366
10.2.7.3 Specification 366
10.2.7.4 Examples 369
10.2.7.5 Enabling the SPARQL Query API 371
10.2.8 SPARQL Update API 372
10.2.8.1 Purpose 372
10.2.8.2 Use Cases 372
10.2.8.3 Specification 373
10.2.8.4 Examples 375
10.2.9 Triple Pattern Fragments 380
10.2.9.1 Background 380
10.2.9.2 Configuration 381
10.2.9.3 Manual Query 383
10.2.9.4 Curl 388
10.2.9.5 Programmatic Access 390
10.2.9.6 References 391
10.3 Application RDF Files 391
10.3.1 ABox, TBox and Configuration Data 391
10.3.2 Content and Configuration Triple Sources 392

10.3.3 Firsttime, Everytime and Filegraph 393
10.3.3.1 Firsttime 393
10.3.3.2 Everytime 394
10.3.3.3 Filegraph 394
10.4 Architecture 395
10.4.1 Overview 395
10.4.2 Vitro 395
10.4.3 VIVO 395
10.4.4 Component View 396
10.4.5 Additional Resources 398
10.4.6 Vitro 398
10.4.7 VIVO and Vitro 398
10.4.8 Software Architecture Overview 399
10.4.8.1 Data 400
10.4.8.2 Logic 401
10.4.8.3 Presentation 402
10.4.8.4 Security 403
10.4.9 The StartupManager 403
10.4.9.1 Overview 403
10.4.9.2 Specifying context listeners 404
10.4.9.3 Writing context listeners 405
10.4.10 VIVO Data Models 406
10.4.10.1 Concepts 406
10.4.10.2 The Data Models 410
10.4.10.3 Increasing complexity 411
10.4.10.4 The ModelAccess class 413
10.4.10.5 Initializing the Models 414
10.4.11 VIVO and the Solr search engine 418
10.4.11.1 What is Solr? 418

10.4.11.2 How does VIVO use Solr? 419
10.4.11.3 How is Solr created and configured? 421
10.4.11.4 The search index 421
10.4.11.5 How does VIVO contact Solr? 425
10.4.12 Image storage 425
10.4.12.1 Access images after changing the default namespace 427
10.4.12.2 How are Images represented in the Model? 427
10.5 Configuration Reference 434
10.5.1 Overview 434
10.5.2 VIVO Runtime Properties 434
10.6 Directories and Files 440
10.6.1 Overview 440
10.6.2 High Level Directories 440
10.6.3 Directory Structure 441
10.7 Freemarker Template Variables and Directives 443
10.8 Graph Reference 443
10.8.1 Overview 443
10.8.2 Listing the graphs used by VIVO 443
10.8.3 The graphs used by VIVO 444
10.8.4 Notes 446
10.9 Ontology Reference 446
10.9.1 Overview 446
10.9.2 Reference Materials 446
10.9.3 Issue Tracking 446
10.9.4 VIVO Ontology Domain Definition 447
10.9.5 Source ontologies for VIVO 448
10.9.5.1 Background 448
10.9.5.2 Ontologies Used in the VIVO Ontology 448
10.9.6 VIVO Classes 448

10.9.6.1 Overview 448
10.9.6.2 Finding the Classes in your VIVO 448
10.9.6.3 VIVO Classes 449
10.9.7 VIVO Object Properties 450
10.9.8 Ontology Diagrams 450
10.9.8.1 Organization Model 451
10.9.8.2 Concept Model 453
10.9.8.3 DateTimeValue and DateTimeInterval Models 454
10.9.8.4 Journal Model 457
10.9.8.5 Person Model 458
10.9.8.6 Teaching Model 459
10.9.8.7 Publication Model 461
10.9.8.8 Grant Model 462
10.9.8.9 Education and Training Model 469
10.9.8.10 Advising Model 471
10.9.8.11 Award Model 473
10.9.8.12 Membership Model 474
10.9.8.13 Ontology Diagram Legend 475
10.9.8.14 Credential Model 475
10.10 Resource Links 476
10.11 Rich export SPARQL queries 478
10.11.1 Rich export SPARQL queries: Address 478
10.11.2 Rich export SPARQL queries: Advising 479
10.11.3 Rich export SPARQL queries: Award 482
10.11.4 Rich export SPARQL queries: Credential 483
10.11.5 Rich export SPARQL queries: Educational Training 485
10.11.6 Rich export SPARQL queries: Funding 487
10.11.7 Rich export SPARQL queries: Membership 489
10.11.8 Rich export SPARQL queries: Outreach 489

10.11.9 Rich export SPARQL queries: Patent 490
10.11.10 Rich export SPARQL queries: Position 492
10.11.11 Rich export SPARQL queries: Presentation 493
10.11.12 Rich export SPARQL queries: Publication 495
10.11.13 Rich export SPARQL queries: Teaching 500
10.12 URL Reference 501
10.12.1 Overview 501
10.12.2 api/datarequest 501
10.12.3 freemarkersamples 501
10.12.4 login 501
10.12.5 logout 501
10.12.6 RecomputeInferences 501
10.12.7 revisionInfo 502
10.12.8 sitemap.xml 502
10.12.9 SiteAdmin 502
10.12.10 SearchIndex 502
10.12.11 tpf/core 502
10.12.12 vivosolr 502
10.13 Utilities Reference 502
10.13.1 jena2tools 503
10.13.1.1 Running Jena2tools 503
10.13.1.2 Help 503
10.13.1.3 Repository 503
10.13.2 jena3tools 503
10.13.2.1 Running Jena2tools 504
10.13.2.2 Help 504
10.13.2.3 Repository 504
11 About This Documentation 505
11.1 Maintaining release-specific info on the Wiki 505

11.1.1 Goals 505
11.1.2 Two types of wiki pages 505
11.1.2.1 Release-specific pages 505
11.1.2.2 Release-neutral pages 505
11.1.3 Approach 506
11.1.3.1 VIVO (main wiki, also known as the project wiki, also known as the community wiki) 506
11.1.3.2 VIVO Release specific wikis, also known as the documentation 506
11.1.3.3 Minimal documentation in the Git repository 506
11.1.4 Between releases 506
11.2 VIVO documentation style guide 506
11.2.1 Page sizes 507
11.2.2 Start with a Table of Contents 507
11.2.3 Use all heading levels 507
11.2.4 Code 508
11.2.5 Linking within the document 508
11.2.6 End with a Children Display macro 508

1 Introduction

- Release Notes (see page 26)
- Functional Overview (see page 30)
- System Requirements (see page 32)

1.1 What is VIVO?

VIVO [Pronunciation: /viːvəʊ/ or *vee-voh*] is member-supported, open source software and an ontology for representing scholarship. VIVO supports recording, editing, searching, browsing, and visualizing scholarly activity. VIVO encourages showcasing the scholarly record, research discovery, expert finding, network analysis, and assessment of research impact. VIVO is easily extended to support additional domains of scholarly activity.

When installed and populated with researcher interests, activities, and accomplishments by an institution, VIVO enables the discovery of research and scholarship across disciplines at that institution and beyond. VIVO supports browsing and a search function which returns faceted results for rapid retrieval of desired information. Content in a VIVO installation may be maintained manually, brought into VIVO in automated ways from local systems of record, such as HR, grants, course, and faculty activity databases, or from database providers such as publication aggregators and funding agencies.

1.2 Release Notes

1.2.1 Overview

VIVO 1.14.0 does not contain VIVO core ontology changes (comparing to VIVO 1.13.0) or require a data migration or modifications to the Solr setup. However, please note that upgrading to VIVO 1.14.0 from a version of VIVO prior to 1.10 requires a triple store unload, use of provided utilities to upgrade, and a reload. See Upgrading VIVO[1]. Moreover, upgrading to VIVO 1.14.0 from a version of VIVO prior to 1.10 requires installation of Solr as a free-standing application. The VIVO 1.14.0 supports two ways of defining user interface labels, via property files (as it was supported in previous versions), and via triplets in the graph in accordance with newly defined UI labels vocabulary (see page 148).

---

[1] https://wiki.lyrasis.org/display/VIVODOC110x/Upgrading+VIVO

1.2.2 What's New

1.2.2.1 Simplified process for installation and customization

The VIVO-languages and Vitro-languages projects are not needed anymore for the installation of VIVO (or Vitro). Customization of UI labels can be specified in local custom ttl files by using private-use language subtags (such as de_DE_x_tib, or fr_CA_x_uqam). Those changes can be made in separated directory in VIVO_HOME, which won't be affected (deleted, overwritten) by any further upgrade of VIVO to newer version.

1.2.2.2 Eliminating language specific FTL files

Phrases sensitive on gender and plural-related transformation of words, and other variations related to other morphological changes in different languages have been replaced with more generic changes insensitive on those linguistic transformations. This eliminates need for some language-specific FTL files, especially in the case of loading data from the graph, and improve user-experience.

1.2.2.3 Simplified process of supporting new languages

Up to VIVO 1.13.0, adding of a new user interface language required changes in VIVO-languages and Vitro-languages projects in different directories for web application property files and for data which should be preserved in graph. Moreover, it required changes of couple Maven specific files (pom.xml), meaning it required a technical person included in the process of adding new language. Starting from the VIVO 1.14.0, adding of new language can be performed by a translator requiring only a basic knowledge of the ontology notation.

1.2.2.4 Improvement of performance for startup of VIVO

The operations of loading and removing sample data, adding and removing model, clearing statements have been improved more than 5 times comparing to VIVO 1.13.0 and prior versions.

1.2.2.5 Faux data properties

Besides support faux object properties, VIVO from 1.14.0 also supports faux data properties.

1.2.2.6 Configuration of usage of Map of Science visualization

A flag visualization.mapOfScience with possible values enable and disable has been introduced in runtime.properties. VIVO adopters/implementers can disable usage of Map of Science.

1.2.3 Resolved Issues

1.2.4 Bug

- [328[2]] - Remove "Other" special case from ChildVClassesWithParent
- [3154[3]] - Term of use page page is in English
- [3175[4]] - Labels on Preferred title deletion form
- [3176[5]] - Labels on Preferred title Edition form
- [3042[6]] - Making Capability Map i18n Compliant
- [3541[7]] - i18n: Error messages not displayed in correct language if locale changed
- [349[8]] - i18n fixes for Processing RDF form
- [3774[9]] - Small typo in a js file
- [3790[10]] - Improved safety for visualisation cahes queries
- [3869[11], 3864[12]] - Some freemarker templates don't have escaped values
- [3850[13]] - Tenderfoot template possibly broken
- [3847[14]] - First class group label always have en-US tag
- [3865[15]] - Faux property participant doesn't work
- [3871[16]] - Upload ontology data that contains blank nodes doesn't work
- [3884[17]] - Some inferred TBox axioms are not removed when the underlying assertions are deleted
- [3889[18]] - JFactTBoxReasonerTest sometimes fail
- [3893[19]] - Language selector doesn't work in Vitro
- [3894[20]] - Autocomplete form for organization name doesn't work

1.2.5 New Feature

- [3760[21]] - Adopting i18n bean to read translations from triple store instead from property files

---

[2] https://github.com/vivo-project/Vitro/pull/328
[3] https://github.com/vivo-project/VIVO/issues/3154
[4] https://github.com/vivo-project/VIVO/issues/3175
[5] https://github.com/vivo-project/VIVO/issues/3176
[6] https://github.com/vivo-project/VIVO/issues/3042
[7] https://github.com/vivo-project/VIVO/issues/3541
[8] https://github.com/vivo-project/Vitro/pull/349
[9] https://github.com/vivo-project/VIVO/issues/3774
[10] https://github.com/vivo-project/VIVO/pull/3790
[11] https://github.com/vivo-project/VIVO/issues/3869
[12] https://github.com/vivo-project/VIVO/issues/3864
[13] https://github.com/vivo-project/VIVO/issues/3850
[14] https://github.com/vivo-project/VIVO/issues/3847
[15] https://github.com/vivo-project/VIVO/issues/3865
[16] https://github.com/vivo-project/VIVO/issues/3871
[17] https://github.com/vivo-project/VIVO/issues/3884
[18] https://github.com/vivo-project/VIVO/issues/3889
[19] https://github.com/vivo-project/VIVO/issues/3893
[20] https://github.com/vivo-project/VIVO/issues/3894
[21] https://github.com/vivo-project/VIVO/issues/3760

- [3800[22]] - Support local extension subtag in the language tag
- [352[23]] - Support for faux data properties
- [356[24]] - Support for combining parameters and apostrophe in UI labels
- [3821[25]] - Enabling/disabling Map of Science visualization

1.2.6 Improvement

- [3792[26]] - Clear statements and remove actions in Manage Jena models menu take too much time
- [3605[27]] - First time loading on HDD with journaling file system can take a lot of time
- [3572[28]] - Get rid of language specific ftl files
- [332[29]] - Fixes for new individual form
- [333[30]] - Move search help page translations to properties
- [3777[31]] - Move qr codes about page translations to n3 configurations
- [3778[32]] - Moving translations for map of science from ftl files to property files
- [334[33]] - Decouple email templates and translations
- [2923[34]] - Replace hard coded text with tags (label keys) in all templates
- [2921[35]] - Support plurals in all languages
- [338[36]] - Language-neutral help text in autocomplete forms
- [3762[37]] - Moving VIVO/Vitro-languages repositories into VIVO/Vitro repositories
- [353[38]] - Adding English as default language
- [3853[39]] - Removing itextpdf dependency
- [3885[40]] - Improved delay when logging in
- [3886[41]] - Added some missed UI labels
- [3896[42]] - Mitigation of Capability map vulnerability

---

[22] https://github.com/vivo-project/VIVO/issues/3800
[23] https://github.com/vivo-project/Vitro/pull/352
[24] https://github.com/vivo-project/Vitro/pull/356
[25] https://github.com/vivo-project/VIVO/issues/3821
[26] https://github.com/vivo-project/VIVO/issues/3792
[27] https://github.com/vivo-project/VIVO/issues/3605
[28] https://github.com/vivo-project/VIVO/issues/3752
[29] https://github.com/vivo-project/Vitro/pull/332
[30] https://github.com/vivo-project/Vitro/pull/333
[31] https://github.com/vivo-project/VIVO/pull/3777
[32] https://github.com/vivo-project/VIVO/pull/3778
[33] https://github.com/vivo-project/Vitro/pull/334
[34] https://github.com/vivo-project/VIVO/issues/2923
[35] https://github.com/vivo-project/VIVO/issues/2921
[36] https://github.com/vivo-project/Vitro/pull/338
[37] https://github.com/vivo-project/VIVO/issues/3762
[38] https://github.com/vivo-project/Vitro/pull/353
[39] https://github.com/vivo-project/VIVO/pull/3853
[40] https://github.com/vivo-project/VIVO/pull/3885
[41] https://github.com/vivo-project/VIVO/pull/3886
[42] https://github.com/vivo-project/VIVO/pull/3896

1.2.7 Release managers

- @Dragan Ivanovic

1.2.8 Acknowledgements

- @Georgy Litvinov (TIB)
- @Brian Lowe (Ontocale)
- @Michel Héon (UQAM)
- @Kevin Day (Texas A&M)
- @William Welling (Texas A&M)
- @Benjamin Kampe (TIB)
- @Mark Vanin (TIB)
- @Anna Guillaumet (SIGMA)
- @Benjamin Gross (Clarivate)
- @Veljko Maksimovic (University of Novi Sad)
- @Ivan Mrsulja (University of Novi Sad)
- @Miloš Popović (University of Novi Sad)
- @Ralph O'Flinn (NIH)
- @Huda Khan (Cornell)
- @Christian Hauschke (TIB)
- @Dragan Ivanovic (LYRASIS / University of Novi Sad)

1.3 Functional Overview

1.3.1 Online Access

VIVO provides an online portal to showcase the academics, their work, and their professional relationships.

1.3.1.1 Linked Open Data

All information within a VIVO system is represented natively in the RDF data model - everything is expressed as subject - predicate - object statements. These statements are written to a triple store, and are made available as RDF documents for each resource, in a number of serialisation formats.

1.3.1.2 Built-in Search

All content is indexed using Solr - a popular open source search platform built on Lucence.

1.3.1.3 Navigation

VIVO provides simple navigation through menus which lead to lists of various types of entities – people, organizations, research. An Index provides access to lists of all types of entities. The Capability Map provides a graphical method for finding people by concepts.

1.3.1.4 Optimisations for Google Indexing

VIVO embeds structured data - hcards and schema.org - into profile pages to better support Google indexing. It also creates a sitemap.xml for all of the profiles in the system, and includes a link to this sitemap in the robots.txt.

It is encouraged that you should register your VIVO instance in Google Webmaster Tools and submit the sitemap.xml for better visibility of how Google is indexing your content.

1.3.1.5 Support for Modern Browsers

VIVO creates standard HTML and CSS that can be used in all modern browsers. Visualisations are mostly built using D3 - a standard JavaScript library - which allows them to be viewed even on mobile devices.

1.3.2 Getting Data into VIVO

1.3.2.1 Manual Data Entry

All screens in VIVO can provide for data entry, for users logged in with sufficient access. There are numerous roles that VIVO provides - from administrators that can edit any of the data in the system, to self editor privileges for users so that they can edit their own profile and related information.

1.3.2.2 Automated Data Entry

It is possible to add data to VIVO using automated tools. VIVO provides a SPARQL update endpoint, which can be used by external tools to manipulate the data, or the VIVO Harvester provides a means to acquire and transform data, and load it directly into the triple store.

1.3.3 Access Control

VIVO has internal storage for user accounts, and can authenticate based on a password (stored as a hash), or via an external authentication mechanism, such as Shibboleth. For externally authenticated users, an internal user account is still required, and is matched based on the external ID.

1.4 System Requirements

1.4.1 Hardware Recommendations

You can install and run VIVO on most modern PC, laptop, or server hardware. Whilst the application layer needs a reasonable amount of memory, the majority of the workload is placed on the storage layers, which as a semantic web application means the triple store. As VIVO aims to be agnostic to the triple store, the precise requirements will depend on your choice of triple store. However, starting from the VIVO 1.12, the default configuration is to use Jena TDB, while VIVO can be optionally configured (see page 334) to use MySQL as a backing store for Jena SDB. TDB is another triple store from the Jena project, but TDB keeps its data in directory of flat files, while SDB uses a relational database. It is recommended that you have very high IO bandwidth for the file system used by Jena TDB or MySQL, and significant memory for caching layers of the database engine.

1.4.1.1 Minimum Specification

2 core x64 processor, 2GB RAM, 100GB HDD

1.4.1.2 Recommended Specification

4 core x64 processor, 16GB RAM, 500GB SSD

Note: If you plan on using the optional Jena SDB database option, I/O performance for MySQL is critical to the responsiveness of the application. Regardless of chosen database, the fastest SSD you can specify will help, as will having direct (e.g. not virtualised) access to it.

1.4.2 Software Requirements

1.4.2.1 Operating System

VIVO is largely agnostic to the OS that it is running on - as a Java application, it is dependent on having a Java Virtual Machine and a Tomcat servlet container. It should be possible to install and run VIVO on any OS where you are able to provide all the other software requirements.

However, most sites will run their installations on a Linux server, and you may find that it is easier to follow the installation instructions on a Linux / UNIX variant. Notably, if you are running Windows, you may need to stop running processes (e.g., Tomcat) to complete some of the instructions, due to file locking semantics on Windows.

1.4.2.1.1 glibc 2.14+ (Linux only)

Linux users, note that one of VIVO's dependencies requires your OS be running glibc[43] 2.14+. All modern Linux variants satisfy this requirement, however some older releases do not. The table below summarizes the minimum release that includes a compatible* version of glibc. The ability to update glibc to 2.14+ on older OS versions varies by distribution.

---

[43] http://www.gnu.org/software/libc/libc.html

Distribution	Ubuntu	Debian	Fedora	CentOS	RHEL	SUSE	Gentoo
Minimum version	12	8	16	7	7	12	12

*This table was created using data from distrowatch.com, accuracy and compatibility not guaranteed. Please leave a comment if you find this table is inaccurate.

1.4.2.2 Java 8 or 11

The minimum requirement is Java 8. Both OpenJDK and Oracle JVMs are compatible. Other JVMs that meet the JDK 8 specification may work, but have not been tested.

OpenJDK 11 is also supported.

Note that you need to have the full Java Development Kit installed in order for Tomcat to operate correctly - the runtime alone is not sufficient.

Warning

Java 9 and 10 have not been tested and are not supported

1.4.2.3 Maven 3.6.1 or later

The installation mechanism uses Maven to package and deploy the VIVO application and other necessary files. Additionally, the development environment also uses Maven to compile the and package the code.

The minimum version of Maven required is 3.6.1, although it is better to use a more recent version of the 3.x releases where possible.

Maven can be downloaded from the following location: http://maven.apache.org/download.html, although you may use a version supplied by your operating system / package manager, providing it meets the minimum requirements.

1.4.2.3.1 Configuring a Proxy

You can configure a proxy to use for some or all of your HTTP requests in Maven. The username and password are only required if your proxy requires basic authentication (note that later releases may support storing your passwords in a secured keystore, in the mean time, please ensure your *settings.xml* file (usually *${user.home}/.m2/settings.xml*) is secured with permissions appropriate for your operating system).

Example:

```
<settings>
  .
  .
  <proxies>
   <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.somewhere.com</host>
      <port>8080</port>
      <username>proxyuser</username>
      <password>somepassword</password>
      <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
    </proxy>
  </proxies>
  .
  .
</settings>
```

1.4.2.4 Tomcat 8 or 9

VIVO is a web application, which requires a servlet engine to host. It has been tested with Tomcat 8.5, and 9. The applications make use of Tomcat context.xml configuration files - if you wish to use an alternative servlet engine, you will need to make the appropriate adjustments.

You may use Tomcat as supplied by your operating system / package manager providing is meets the minimum requirements, or you can download it from: http://tomcat.apache.org/download-85.cgi[44]. Note, VIVO does not currently support Tomcat 10+.

Tomcat User: When running, Tomcat is usually launched under an unprivileged user account. As VIVO needs to be able to read and write to the home directory, you must ensure that permissions are set on the home directory correctly. This is most easily achieved by assigning ownership to the user that Tomcat is running as.

1.4.2.5 Optional - MySQL / MariaDB 5.5 or later (or any other supported by Jena SDB)

VIVO can optionally be configured to use a legacy database called Jena SDB, which requires an SQL database to operate. This may be an attractive option for system administrators familiar with SQL databases, however note that Jena SDB was 'retired' by the Apache Jena project in 2020.

Once MySQL is installed, you only need to create a user and schema - see Jena SDB and MySQL setup (see page 334). VIVO will create the necessary tables and load the default data on startup.

Alternative databases: Jena SDB supports other databases - including PostgreSQL and Oracle. If you wish to use a different database, you will need to add the appropriate Java libraries to the application, and configure the VitroConnection.DataSource.* settings in runtime.properties so that Jena knows what database it is operating with.

---

[44] http://tomcat.apache.org/download-80.cgi

1.4.3 Running VIVO behind an Apache server

Most sites choose to configure their VIVO system with an Apache HTTPD web server to accept requests and then proxy them to the VIVO Tomcat context. This will make Vitro available at `http://example.com` instead of `http://example.com:8080/vitro`. It will also allow the use of external authentication.

Setup HTTPD to send all of the requests that it receives to Tomcat's AJP connector. This can be done in HTTPD 2.x with a simple directive in httpd.conf:

```
ProxyPass / ajp://localhost:8009/
```

There are at least 2 ways to do this, along with a 3rd method that hasn't been documented yet.

1. In the tomcat webapps directory, create a symbolic link called ROOT to vivo. This will redirect all requests to root to the vivo application.
   eg:

   ```
   cd /usr/local/tomcat/webapps
   rm -rf ROOT
   ln -s vivo ROOT
   ls -latr
   vivosolr
   vivo
   ROOT -> vivo
   ```

2. Modify the <Host> in Tomcat server.xml (located in `[tomcat root]/conf/`) so that the context path is empty to allow VIVO to be served from the root path. Locate the `<Host name="localhost"...>` directive and update as follows:

```
<Host name="localhost" appBase="webapps"
        DeployOnStartup="false"
        unpackWARs="true" autoDeploy="false"
        xmlValidation="false" xmlNamespaceAware="false">

    <Alias>example.com</Alias>

        <Context path=""
                docBase="/usr/local/tomcat/webapps/vitro"
                reloadable="true"
                cookies="true" >

            <Manager pathname="" />
```

```
    </Context>

    ...
```

After setting up the above, it is recommended that you modify the Tomcat AJP connector parameters in server.xml. Look for the <connector> directive and add the following properties:

```
connectionTimeout="20000" maxThreads="320" keepAliveTimeout="20000"
```

Note: the value for maxThreads (320) is equal or greater than the value for MaxClients in the apache's `httpd.conf` file.

3. in the tomcat/conf directory - create a ROOT.xml with the context element as stated above. This was suggested, but somebody needs to verify and complete this document.

2 Installing VIVO

Be sure to review System Requirements (see page 32) before installing VIVO.

2.1 Installing from Distribution

2.1.1 Overview

Download the distribution from the VIVO repository on [45]GitHub. The standard distribution consists of the projects required to create a home directory for VIVO, and to copy the web application and search index. All the compiled code and dependencies are resolved from the Maven central repository at the time you run Maven.

The standard distribution is laid out as follows:

```
vivo-1.14.0/
  pom.xml
  example-settings.xml
  home/
    pom.xml
    src
  webapp/
    pom.xml
    src
```

2.1.2 Preparing the Installation Settings

In order to fully install VIVO, you need to create a settings file that provides some essential information:

app-name

vivo-dir

tomcat-dir

This file needs to be created following the Maven Settings Reference[46]. A template file already exists within the VIVO standard distribution, called "example-settings.xml". You may copy this file (it can be called anything you like), and edit the contents to fit your requirements / system configuration.

---

[45] https://github.com/vivo-project/VIVO/releases
[46] https://maven.apache.org/settings.html

2.1.3 Permissions

Make sure:

- The maven user has write permission to the Tomcat webapps directory. Maven will fail silently if it cannot copy files to tomcat.
- The tomcat user has write permission to the <vivo-dir>/home directory. Please note there might be some specificity for tomcat read/write permissions depending on operating system and tomcat version, such as Tomcat 9 (Debian)[47].

2.1.4 Installing VIVO

Once you have an appropriate settings file (these instructions will assume that you are using example-settings.xml - replace this with your actual file), you simply need to run Maven, specifying the install goal and your settings file.

```
$ cd VIVO
VIVO$ mvn install -s example-settings.xml
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] VIVO Installer
[INFO] VIVO Prepare Home
[INFO] VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat.

In order to run VIVO, please read the section below "Completing the Installation (see page 41)".

---

[47] https://salsa.debian.org/java-team/tomcat9/blob/master/debian/README.Debian#L38

2.2 Installing from GitHub

2.2.1 Preparing the Repositories

In order to install the development code from GitHub, you need to clone both the Vitro and VIVO repositories from the vivo-project organization. These clones should be in sibling directories called "Vitro" and "VIVO" respectively. These will all point to the 1.14.1-SNAPSHOT:

```
$ git clone https://github.com/vivo-project/Vitro.git Vitro -b rel-1.14-maint
$ git clone https://github.com/vivo-project/VIVO.git VIVO -b rel-1.14-maint
drwxr-xr-x  user  group  1 Dec 12:00  Vitro
drwxr-xr-x  user  group  1 Dec 12:00  VIVO
```

If you do not place the Vitro code in a sibling directory called "Vitro", then you will have to supply the "vitro-core" property to Maven - e.g. mvn package -Dvitro-core=~/Vitro
It is expected that the Maven project numbers are kept in sync between the Vitro / VIVO projects, however, depending on when you update / sync your repositories, you may need to adjust the project version numbers for the build to work.

2.2.2 Preparing the Installation Settings

In order to fully install VIVO, you need to create a settings file that provides some essential information:

app-name

vivo-dir

tomcat-dir

This file needs to be created following the Maven Settings Reference[48]. A template file already exists in the "installer" directory within the VIVO project, called "example-settings.xml". You may copy this file (it can be called anything you like) and edit the contents to fit your requirements / system configuration.

2.2.3 Permissions

Make sure:

- The maven user has write permission to the Tomcat webapps directory. Maven will fail silently if it cannot copy files to tomcat.
- The tomcat user has write permission to the <vivo-dir>/home directory.

---

[48] https://maven.apache.org/settings.html

2.2.4 Installing VIVO

2.2.4.1 Default Installer

Once you have an appropriate settings file (these instructions will assume that you are using installer/example-settings.xml - replace this with your actual file), you simply need to run Maven, specifying the install goal and your settings file.

```
$ cd VIVO
VIVO$ mvn install -s installer/example-settings.xml
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] VIVO Installer
[INFO] VIVO Prepare Home
[INFO] VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat. To run VIVO, please read the section below "Completing the Installation (see page 41)".

2.2.4.2 Custom Installer

If you want to use the source code / GitHub clone with your own customizations, you can exclude the supplied installer project, and use your own customized installer project instead. To do so, you need to supply the location of your custom installer project as the "vivo-installer-dir" property. This can be done on the command line or in the settings.xml. If you are supplying a relative path, it should be relative to the location of the VIVO/pom.xml.

```
$ cd VIVO
VIVO$ mvn install -s installer/example-settings.xml -Dvivo-installer-dir=../myedu-vivo
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] Custom VIVO Installer
[INFO] Custom VIVO Prepare Home
[INFO] Custom VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat, including any customizations that are defined in your local installer project. To run VIVO, please read the section below "Completing the Installation (see page 41)".

2.3 Completing the Installation

2.3.1 Configure the Home Directory

There are two configuration files that are required to be in the home directory. By default, the installer does not create them so that they are not overwritten when you redeploy the application. Instead, example files are created in the home directory, which can be copied and used as the basis for your installation.

```
$ cd /usr/local/vivo/home/config (When you install vivo for the first time
applicationSetup and runtime files are in home/src/main/resources/config)
/usr/local/vivo/home/config$ cp example.runtime.properties runtime.properties
/usr/local/vivo/home/config$ cp example.applicationSetup.n3 applicationSetup.n3
```

2.3.2 Configure the Database Schema

Optionally, you may choose to use the older Jena SDB database backed by MySQL. SDB is not recommended for new installations, however it may offer some situation-dependent advantages for some users. For more information on SDB setup, see Jena SDB and MySQL setup (see page 334).

2.3.3 Configure and Start Solr

Solr must be independently deployed and configured with the schema expected by VIVO. VIVO must then be configured to connect to the external Solr.

1. Download and install the latest 8.x version of Solr[49] (installation instructions[50])
    a. The directory in which Solr is installed is referenced below as ${SOLR_HOME} (e.g., /opt/solr/solr-8.11.1)
2. Add the vivocore[51] directory of the vivo-solr GitHub repository and its contents into ${SOLR_HOME}/server/solr
    a. The end result should be a directory structure such as:

```
${SOLR_HOME}/server/solr/vivocore/core.properties
                                └── conf/
                                    ├── currency.xml
                                    ├── elevate.xml
                                    └── ...
```

3. Start Solr

```
${SOLR_HOME}/bin/solr start
```

4. Remove schema.xml from ${SOLR_HOME}/server/solr/vivocore/conf
    a. When solr was started it created the managed-schema automatically from the schema.xml and is no longer needed
5. Update VIVO runtime.properties[52] as below to point to the URL of your Solr

```
vitro.local.solr.url = http://localhost:8983/solr/vivocore
```

6. Start VIVO!
    a. Note: If VIVO was started before connecting to Solr, please restart VIVO.

2.3.4 Configure and Start Tomcat

2.3.4.1 Set JVM parameters

VIVO copies small sections of your RDF database into memory in order to serve Web requests quickly (the in-memory copy and the underlying database are kept in synch as edits are performed).

---

[49] http://archive.apache.org/dist/lucene/solr/8.11.1/
[50] https://lucene.apache.org/solr/guide/8_11/installing-solr.html
[51] https://github.com/vivo-project/vivo-solr/tree/vivo-solr-1.11.0/vivocore
[52] https://github.com/vivo-project/VIVO/blob/develop/home/src/main/resources/config/example.runtime.properties

VIVO may require more memory than allocated to Tomcat by default. With most installations of Tomcat, the setenv.sh or setenv.bat file in Tomcat's bin directory is a convenient place to set the memory parameters. *If this file does not exist in Tomcat's bin directory, you can create it.*

For example:

```
export CATALINA_OPTS="-Xms512m -Xmx512m -XX:MaxPermSize=128m"
```

This tells Tomcat to allocate an initial heap of 512 megabytes, a maximum heap of 512 megabytes, and a PermGen space of 128 megs. Larger values may be required, especially for production installations in large enterprises. In general, VIVO runs more quickly if given more memory.

If an OutOfMemoryError occurs during VIVO execution, increase the heap parameters and restart Tomcat.

2.3.4.2 Set security limits

VIVO is a multithreaded web application that may require more threads than are permitted under the default configuration of your operating system. Ensure that your installation can support the required number of threads for your application. For a Linux production environment, you may wish to make the following edits to /etc/security/limits.conf, replacing apache and tomcat with the appropriate user or group name for your setup:

```
apache hard nproc 400
tomcat hard nproc 1500
```

2.3.4.3 Set URI encoding

In order for VIVO to correctly handle international characters, you must configure Tomcat to conform to the URI standard by accepting percent-encoded UTF-8.

Edit Tomcat's conf/server.xml and add the following attribute to each of the Connector elements: URIEncoding="UTF-8".

```
<Server ...>
  <Service ...>
    <Connector ... URIEncoding="UTF-8"/>
      ...
    </Connector>
  </Service>
</Server>
```

Some versions of Tomcat already include this attribute as the default.

2.3.4.4 **Take care when creating Context elements**

The webapp (VIVO) in the VIVO distribution includes a "context fragment" file, containing some of the deployment information for that webapp.

Tomcat allows you to override these context fragments by adding Context elements to server.xml. If you decide to do this, be sure that your new Context element includes the necessary deployment parameters from the overridden context fragment.

2.3.4.5 Starting Tomcat

If everything has been completed successfully, then you should simply be able to start Tomcat at this point, and VIVO will be available. If you are using a Tomcat supplied by your operating system / package manager, then use your normal means for starting the application server.

Otherwise, start Tomcat by running the startup script - e.g.

```
$ /usr/local/tomcat/bin/startup.sh
```

2.4 Verify Your Installation

If you have completed the previous steps, you have good indications that the installation was successful.

- When you Start tomcat, you see that Tomcat recognizes the webapp, and that the webapp is able to present the initial page.
- The startup status will indicate if the basic configuration of the system was successful. If there were any serious errors, you will see the status screen and will not be allowed to continue with VIVO. If there are warnings, you will see the status screen when you first access VIVO, but after that you may use VIVO without hinderance. In this case, you can review the startup status from **siteAdmin -> Startup status**.
- Log in as root. Your root username is vivo_root@mydomain.edu (or the email you configured in runtime.properties). The first-time root password is rootPassword. You will be asked to change it.

Here is a simple test to see whether the ontology files were loaded:

- Click on the "Index" link on the upper right, below the logo. You should see a "locations" section, with links for "Country" and "Geographic Location." The index is built in a background thread, so on your first login you may see an empty index instead. Refresh the page periodically to see whether the index will be populated. This may take some time: with VIVO installed on a modest laptop computer, loading the ontology files and building the index took more than 5 minutes from the time that Tomcat was started.
- Click on the "Country" link. You should see an alphabetical list of the countries of the world.

Here is a test to see whether your system is configured to serve linked data:

- Point your browser to the home page of your website and click the "Log in" link near the upper right corner. Log in with the rootUser.emailAddress you set in `runtime.properties` . If this is your first time logging in, you will be prompted to change the password.
- After you have successfully logged in, click "site admin" in the upper right corner. In the drop down under "Data Input" select "Faculty Member(core)" and click the "Add individual of this class" button.
- Enter the name "test individual" under the field "Individual Name," scroll to the bottom, and click "Create New Record." You will be taken to the "Individual Control Panel." Make note of the value of the field "URI" - it will be used in the next step.
- Open a new web browser or browser tab to the page http://lodview.it/. Enter the URI of the individual you created in the previous step and click "go."
- In the resulting page search for the URI of the "test individual." The page should display information for the individual, including a rdfs: label and rdf:type. This indicates that you are successfully serving linked RDF data. If the service returns an error, you are not successfully serving linked data.

Finally, test the search index.

- Type the word "Australia" into the search box and click on the Search button. You should see a page of results, with links to countries that border Australia, individuals that include Australia, and to Australia itself. To trigger a rebuild of the search index, you can log in as a site administrator and go to **Site Admin -> Rebuild search index**.

3 Upgrading VIVO

If you are already running VIVO 1.10.x - 1.13.x, you can upgrade to VIVO 1.14.x without making any configuration or data changes. Follow the procedure in Installing VIVO (see page 37) while using your existing installation settings (installation.xml). If you have customized your VIVO 1.13 installation, you will need to take action to preserve those customizations.

If you have customizations, please see:

- Preserving Customizations During Build (see page 51) for processes to include your customizations in a VIVO build
- Additional Considerations (see page 46) for notes which may impact your customizations

3.1 Upgrading from VIVO 1.10.x prior versions

If you are running VIVO 1.9.x please perform steps needed to upgrade to 1.10.x at Upgrading VIVO 1.9.x to 1.10.x[53]

If you are running VIVO 1.8.x please perform steps needed to upgrade to 1.9.x at Upgrading VIVO 1.8.x to 1.9.x[54]

If you are running VIVO 1.5.x - 1.7.x please perform steps needed to upgrade to 1.8.1 at Upgrading VIVO from release 1.5 to release 1.8.1[55]

3.2 Additional Considerations

3.2.1 Overview

If your site does not have customizations, and does not use Jena-based applications, these considerations do not apply.

---

[53] https://wiki.lyrasis.org/display/VIVODOC110x/Upgrading+VIVO
[54] https://wiki.lyrasis.org/display/VIVODOC19x/Upgrading+VIVO
[55] https://wiki.lyrasis.org/display/VIVODOC18x/Upgrading+VIVO+from+release+1.5+to+release+1.8.1

3.2.2 Upgrade Local Java Code Using Jena

If you have local customisations or additional applications that make use of the Jena libraries, you will need to upgrade these to work with Jena 3. Mostly this is simply a case of renaming any packages in imports for Jena classes:

import com.hp.hpl.jena.*

becomes

import org.apache.jena.*

However, some classes have been moved, or removed, and some interfaces have additional methods. So in rare cases you may find that you need to make a few small changes beyond this.

3.2.3 Notes on Other Dependency Changes

To remove the possibility of incompatible classes being loaded, and to remove known vulnerabilities from the code base, most of the Java dependencies in VIVO have been removed, updated or replaced.

For the most part, this will have minimal impact on local customisations.

Some libraries - such as commons-lang3 - have new package names, but mostly compatible classes, and usually just require the imports to be adjusted.

The CSV libary was outdated and unmaintainted, and has been replaced with commons-csv.

There were originally 5 JSON libaries (Jackson, Gson, Glassfish, Sourceforge.net[56] and Json.org[57] parsers). All code is now using Jackson, and the other parsers have been removed.

38 dependencies have been removed from the standard distribution. If you have any customisations that make use of them, that should work, but you will need to add the dependencies to your own projects.

22 dependencies will appear to have been added, but these are mostly from unbundling the owlapi OSGi dependency, and these - along with other conflicting classes - had been packaged as part of the bundle.

while every effort has been made to eliminate known vulnerabilities, the Maven dependency-check plugin still reports 13 vulnerabilities across 8 libraries - for which these are currently the latest releases.

In total, 11 out of an original 113 dependencies are still at the same version as the previous release.

3.2.4 UI Changes

3.2.4.1 jQuery 1.12.4

Bootstrap based themes require a newer version of jQuery than was shipped with VIVO. In order for all of the functionality aross the UI to work, and to minimise the amount of duplication in the UI, it was necessary to upgrade all of the pages and plugins that used jQuery, so that the same upgraded version works across all of the pages and themes.

---

[56] http://Sourceforge.net

[57] http://Json.org

This release does require some migration of javascript that makes use of it. There is a script - jquery-migrate.js that helps with this, providing extra backwards compatibility, and logging warnings to the console whenever an old method is used.

All of the existing code that was relying on the migrate script has been updated, so that it is no longer needed.

To aid transition, VIVO still ships with the jquery-migrate.js script, allowing most existing code to work. You should monitor the Javascript console, and apply updates if you see any logged messages in your customisations - future versions of VIVO will remove this migration script in order to upgrade to later versions.

3.2.4.2 jQuery plugins

In order to support the upgrade to jQuery 1.12.4, and remove any backward compatibility messages from the Javascript console, the following plugins have been upgraded:

jQuery UI

jCrop

qTip

DataTable

In addition, the following plugins have been replaced with equivalents for compatibility and/or licensing issues:

Old	New	Notes
mb.FlipText	Jangle	Used for rotating text on the graphs (e.g. temporal graph)
isotope	wookmark	Used to render the three columns on the index page

3.2.4.3 D3 v4

This is another major upgrade that has architectural changes to improve modularity and make writing visualizations easier, as well as a selection of new features.

Where D3 was being included by VIVO (e.g. on the profile page, in co-authorship networks, etc.), these now include D3 v4, and the visualizations have been upgraded to use D3 v4.

The only visualization that has NOT been upgraded to D3 v4, is the Capability Map - as a full page visualization, that page is still including it's own D3 v3.

When upgraading, you have a few choices:

1) If you have a full page custom visualization, you can continue to specify whatever libraries and versions that you want to use, although if you are referencing the "shared" D3, you will need to adjust this to include your own D3 that is the version you require.

2) If you are embedding the visualization on a page that may have other visualizations, you should either:

a) Upgrade your custom visualization to use D3 v4.

b) Render your visualization into an iframe, so that you can specify your own javascript dependencies.

By making this change now, we are getting on to a maintained version of D3 (v3 has not had any releases for 18 months), and it prepares us for using D3.express[58] in the future, for more dynamic visualization building.

3.2.5

List View Configurations

To produce complex views of data associated with properties (e.g. a person's publication list), VIVO allows the association of externalized SPARQL queries, and associated Freemarker template links.

The configuration for this are the files called "listViewConfig-*" in the <webapp>/configs/ directory.

Due to the performance of OPTIONAL clauses for some triple stores, in VIVO 1.x the configuration files usually consist of a SELECT query (to retrieve the data for the associated Freemarker template), and one or more CONSTRUCTs (using UNIONs) to create a smaller model that the SELECT query would then be performed against.

This creates additional buffering of an intermediate model, the overhead of performing multiple queries, and makes the configurations harder to write and maintain.

For triple stores with poor OPTIONAL performance, this is a result of the OPTIONAL clause returning large amounts of data that is then joined to the rest of the query.

This can be avoided by making the OPTIONAL clauses contain the restrictions that they will be joined to, in addition to them appearing outside for the join.

As this is not necessary for all triple stores, an element <precise-subquery> has been introduced, so that these additional statements can be filtered out for triple stores where they aren't required.

So,

```
SELECT ?menuItem
       ?linkText
 WHERE {
    ?subject ?property ?menuItem .
    OPTIONAL {
        ?menuItem display:linkText ?linkText .
    }
}
```

can be rewritten as

```
SELECT ?menuItem
```

---

[58] https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0

```
       ?linkText
 WHERE {
    ?subject ?property ?menuItem .
    OPTIONAL {
        <precise-subquery>?subject ?property ?menuItem .</precise-subquery>
        ?menuItem display:linkText ?linkText .
    }
}
```

When all OPTIONAL clauses are rewritten like this, the <query-construct> elements can be removed, for approximately 10% performance improvement, and a larger reduction in Java processing overhead. This allows the web server to scale to handle more requests.

If you have customized listViewConfig-*.xml files, you do not need to rewrite them for VIVO 1.10 - they will work unmodified. However, you will have a small performance improvement, and more readable, maintainable queries, if you choose to modify them to take advantage of the new features.

3.2.6 Servlet 3.0 Upgrade

The web.xml that ships with VIVO has been updated to use 3.0 semantics (the required version of Tomcat already supported 3.0).

If you have customisations that introduce new servlets, then you can still add the configuration to web.xml, or you can modify the servlet to use @WebServlet annotations.

If you are replacing a servlet, then you will need to map the existing servlet to an unused url, and map the new servlet by adding configuration for these servlets to web.xml.

Alternatively, if you want to disable a servlet, then you can set the web.xml back to 2.5 spec, and add all of the servlet configuration explicitly to the web.xml.

3.2.7 Java Dependencies

3.2.7.1 HttpClient

Due to OSGi bundling of some of the core dependencies, VIVO had been shipping three different versions of HttpClient, all of which were incompatible with each other, and sometimes generated errors depending on which code paths got executed first.

VIVO 1.10 brings convergence around HttpClient 4.5.3, removing the problems caused previously. If you have any customisations that depend on HttpClient (or the fluent api - fluent-hc), please ensure that you are using the versions that are already indlueded with VIVO. This may require some minor adjustments to your client code to make it compatible.

3.2.7.2 OSGi Dependencies

The OWLAPI previously included with VIVO was an OSGi bundle, and included the classes of a number of libraries (such as HttpClient above). This causes classloading conflicts. In VIVO 1.10, we are depending directly on the libraries that were being bundled, rather than the bundle in it's entirety.

In addition, Jena has OSGi dependencies for HttpClient, leading to more duplicate classes. These have been explicitly excluded - see dependencies/pom.xml[59] for how this is done.

3.2.7.3 JSON Parsers

Four JSON parsers - GSON (com.google.gson), Glassfish (javax.json.Json, Sourceforge.net[60] (net.sf.json) and JSON.org[61] (org.json) have been removed, to simplify the code base, reduce vulnerabilities and improve performance.

All JSON parsing in VIVO is now handled through Jackson.

If you have local customisations that use a different JSON parser, you can add the dependency to your projects, although it is recommended that migrating to Jackson is preferable for long term maintenance.

3.2.7.4 Replaced Dependencies

The CSV parser has been replaced with commons-csv.

3.2.7.5 Removed Dependencies

The following dependencies have been removed from VIVO. If you have customisations that require any of them, you will need to add them as dependencies in your own projects.

agrovocws-3.0.jar, asm-3.1.jar, aterm-java-1.8.2.jar, axis-1.3.jar, axis-jaxrpc-1.3.jar, axis-saaj-1.3.jar, bcmail-jdk14-1.38.jar, bcprov-jdk14-1.38.jar, bctsp-jdk14-1.38.jar, c3p0-0.9.2-pre4.jar, cglib-2.2.jarcommons-beanutils-1.7.0.jar, commons-discovery-0.2.jar, cos-05Nov2002.jar, csv-1.0.jar, cxf-xjc-runtime-2.6.2.jar, cxf-xjc-ts-2.6.2.jar, dom4j-1.6.1.jar, ezmorph-1.0.4.jar, gson-2.5.jar, jai_codec-1.1.3.jar, jai_core-1.1.3.jar, JavaEWAH-0.8.6.jar, javax.json-api-1.0.jar, jjtraveler-0.6.jar, jsonld-java-jena-0.2.jar, json-lib-2.2.2-jdk15.jar, lucene-analyzers-common-5.3.1.jar, lucene-core-5.3.1.jar, lucene-memory-5.3.1.jar, lucene-queries-5.3.1.jar, lucene-queryparser-5.3.1.jar, lucene-sandbox-5.3.1.jar, mail-1.4.jar, mchange-commons-java-0.2.2.jar, shared-objects-1.4.9.jar, sparqltag-1.0.jar, stax-api-1.0-2.jar, wsdl4j-1.5.1.jar, itextpdf-5.5.12.jar, icu4j-59.1.jar, spring-beans-5.3.18.jar, spring-context-5.3.18.jar

3.3 Preserving Customizations During Build

3.3.1 Overview

VIVO can be customized in many ways. Most sites customize the interface using themes. See Creating a custom theme (see page 210). Many sites create ontology extensions to introduce local terminology. See Adding Additional Ontologies to VIVO (see page 262). Sites can add custom configurations for the Data Distribution API (see Data Distribution API (see page 339)), providing custom APIs returning data you specify in formats you specify at web addresses you specify.

In each case, files distributed with VIVO are replaced or added. The process described here uses a Maven custom installer to replace and add your files to those distributed with VIVO.

---

[59] https://github.com/vivo-project/Vitro/blob/develop/dependencies/pom.xml
[60] http://Sourceforge.net
[61] http://JSON.org

3.3.2 Maven Custom Installer

3.3.3 Example – Custom Theme

3.3.4 Example – Local Ontology

3.3.5 Example – Data Distribution API

4 Exploring VIVO

- Logging in to VIVO (see page 53)
- Sample Data (see page 54)
- Restoring VIVO to First Time State (see page 61)

4.1 Overview

As a first time VIVO user, you should take some time to familiarize yourself with the interface, learn how VIVO stores data, consider options for configuring VIVO, and learn how to create user accounts in VIVO.

When VIVO starts up the very first time, it will notice that its data store is empty, and will proceed to load data from `firsttime` directories. Data in these directories include the ontologies to be used by VIVO. These data are loaded exactly once. VIVO will then proceed to load data in the `everytime` and `filegraph` directories. Anytime VIVO is started after the first time, it will check for data in the `everytime` and `filegraph` directories and compare that data with the data in its data store and update the data store as necessary.

In this section we provide an exercise regarding sample data. You can load the sample data into your VIVO, explore the interface, and learn how the sample data is stored. When you are finished with the sample data, you can restore your VIVO to its "firsttime" state. From there you can review the configuration options, choose options best suited for your VIVO, and then create user accounts. You will then be ready to begin loading your data into your VIVO, providing your institution with data for finding experts and representing their scholarship.

4.2 Logging in to VIVO

To log into VIVO using the web browser, navigate to your institution’s instance of VIVO.

- Click the "Log in" link near the upper right corner.
- Enter your username (usually email or external authentication ID) and your password (see note below)
- Click the “Log in” button and you will be redirected to the Home page.

Note: *If you have not yet created any user accounts in VIVO, you can log in as the root user that you set up in the configuration file (rootUser.emailAddress in runtime.properties). If this is your first time logging in, the password will be "rootPassword". You will be required to set a new password to complete the login process.*

4.3 Sample Data

4.3.1 Overview

VIVO has many features and can display many kinds of data related to the scholarship of the individuals at your institution. VIVO provides sample data which can demonstrate the features of VIVO, and help familiarize you with the formats VIVO uses to store its data. The sample data includes a fictional university, fictional departments, fictional faculty members and collaborators, fictional publications and grants, memberships and other elements of scholarship common in VIVO. The sample data does not include an example of every type of thing that can be stored in VIVO, nor does it contain many faculty. It is intended to demonstrate the most common elements of VIVO.

4.3.2 Preparing Your VIVO

To use the sample data, and follow the examples here regarding the sample data, you will want your VIVO to contain only the sample data. Do not add the sample data to your data. VIVO uses data from a database you specify. We will create a database for the sample data, and tell VIVO to use that database when loading and using the sample data. At the end of this page, we will tell VIVO to use the database that you used when you installed VIVO. In this way, your data, and the sample data will always be separate.

In the steps that follow, we assume that you have installed VIVO according to the installation instructions, and that MySQL and Tomcat are running. If this is not the case, please complete the installation and test it, before attempting to use the instructions here.

To create a database for the sample data, and tell VIVO to use it, follow the steps below:

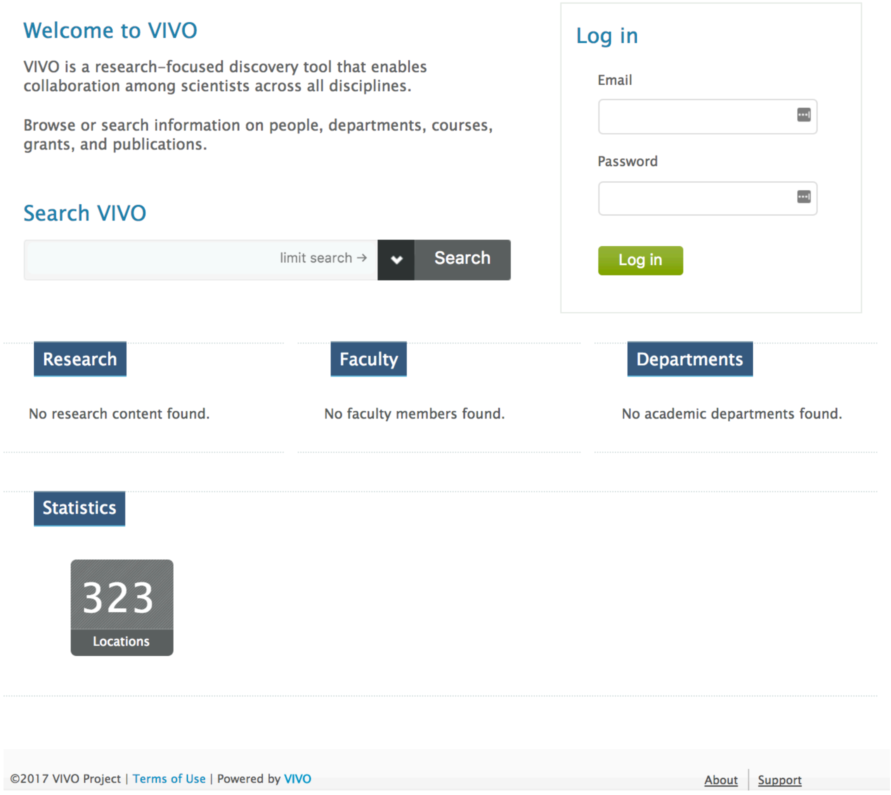
1. Record the name of the database you used to install VIVO. In the installation instructions, this is referred to as `vitrodb`, but you may have named it something else. After you have finished with the sample data, you will follow steps to reset VIVO to use this database.
2. Create a new database in MySQL

Create sample database

```
$ mysql -u root -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.9 MySQL Community Server (GPL)
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE sampledb CHARACTER SET utf8;
mysql> GRANT ALL ON sampledb.* TO 'vitrodbUsername'@'localhost' IDENTIFIED BY
'vitrodbPassword';
```

3. Edit the `runtime.properties` file to tell VIVO the name of the database it should use. In this case `sampledb`.
4. Restart Tomcat

On completing these steps, your VIVO will be using an "empty" database. You may need to wait for VIVO to start the first time, as VIVO automatically loads data from files distributed with VIVO. You may ned to refresh your browser. You may need to wait will VIVO indexes the first time data. It may take several minutes for VIVO to restart, load the first time data, and index it. When VIVO has completed its work, and you have refreshed your browser, your home page should look like:



Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

No research content found.

No faculty members found.

No academic departments found.

©2017 VIVO Project | Terms of Use | Powered by VIVO

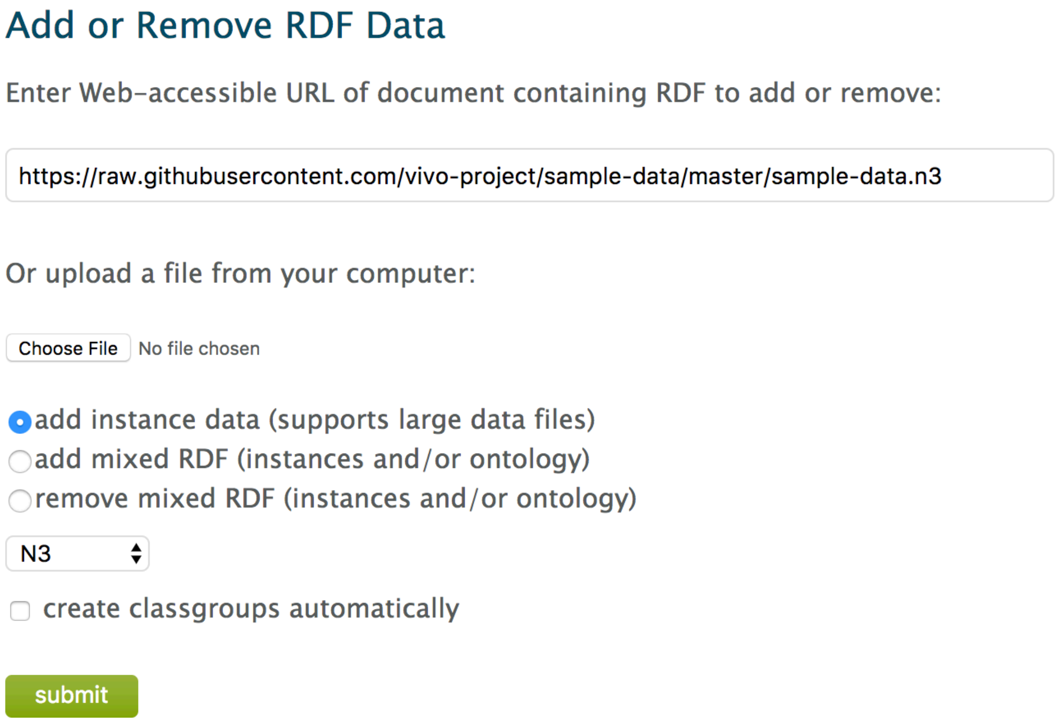
You are are now ready to load the sample data.

4.3.3 Loading the Sample Data

To load the sample data, follow the steps below.

1. Log in to your VIVO as a site admin
2. Goto to Site Admin / Add/Remove RDF Data

3. Enter the address of the sample data, https://raw.githubusercontent.com/vivo-project/sample-data/master/sample-data.n3, select "add instance data," and set the file type to "N3." See below



Add or Remove RDF Data

Enter Web-accessible URL of document containing RDF to add or remove:

https://raw.githubusercontent.com/vivo-project/sample-data/master/sample-data.n3

Or upload a file from your computer:

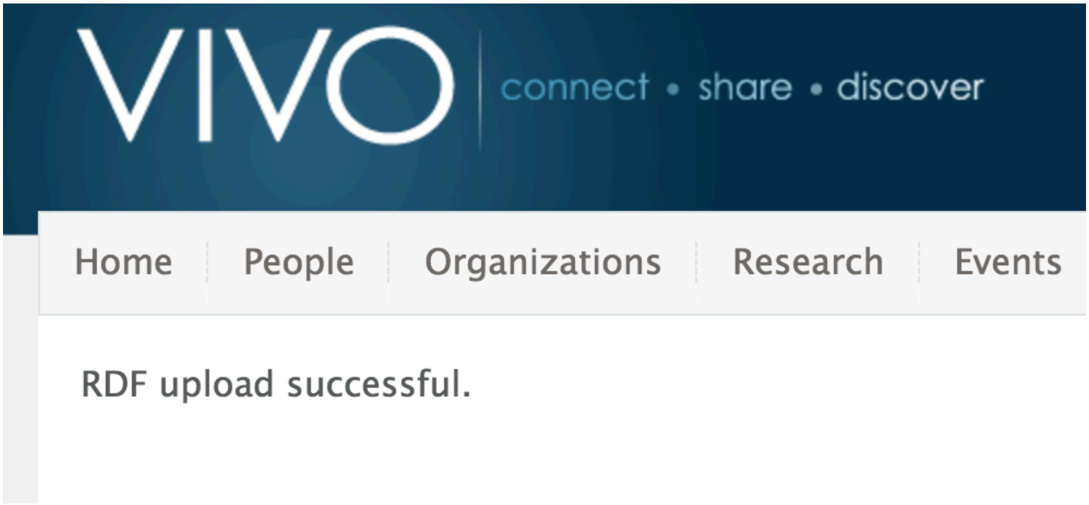
Choose File No file chosen

● add instance data (supports large data files)
○ add mixed RDF (instances and/or ontology)
○ remove mixed RDF (instances and/or ontology)

N3

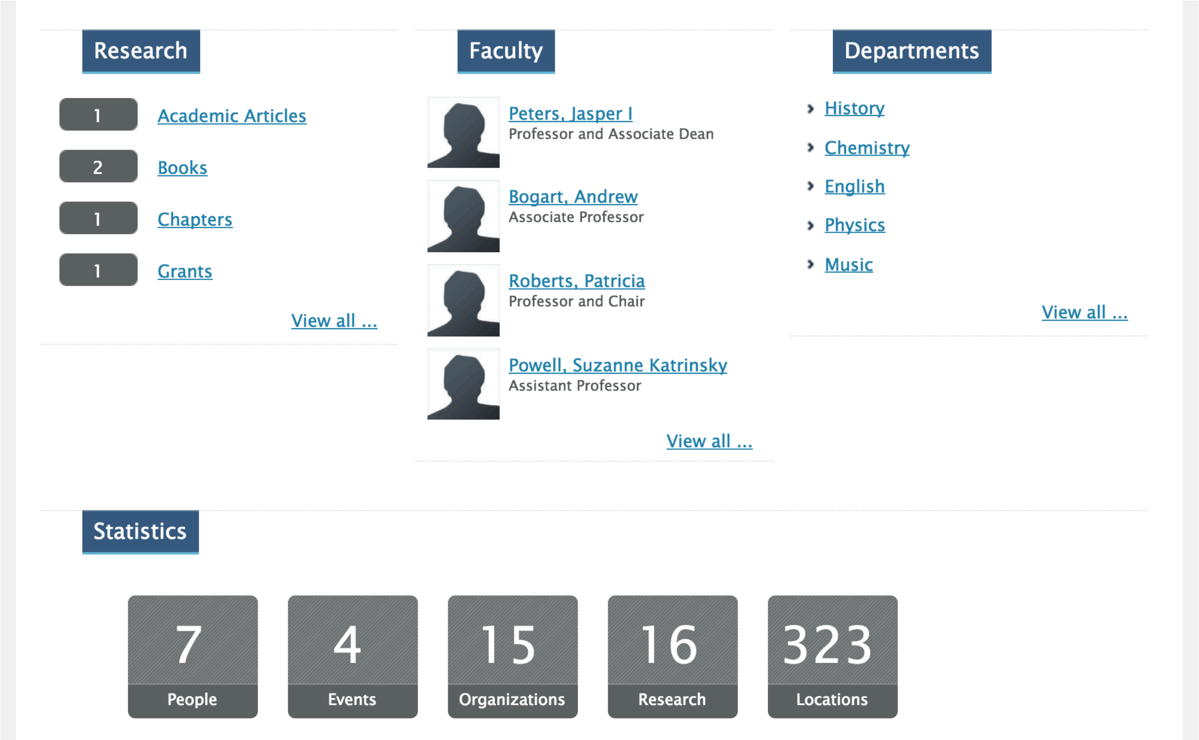
☐ create classgroups automatically

4. Check to make sure you have the form filled out properly: 1) the URL for the sample data has been entered as shown; 2) "add instance" is selected; 3) "N3" is selected as the file type. Press Submit.
5. After a brief upload, you will see



RDF upload successful.

6. Navigate to your home page. You should see


Research
1
Academic Articles
2
Books
1
Chapters
1
Grants
View all ...
Faculty
Peters, Jasper I
Professor and Associate Dean
Bogart, Andrew
Associate Professor
Roberts, Patricia
Professor and Chair
Powell, Suzanne Katrinsky
Assistant Professor
View all ...
Departments
History
Chemistry
English
Physics
Music
View all ...
Statistics
7
People
4
Events
15
Organizations
16
Research
323
Locations

7. That's it! Let's start exploring.

4.3.4 Exploring the Interface

Use the interface to search for terms. You may wish to try "Chemistry" or "Derrida" or "Roberts".

Sign on as a site administrator and use the interface. You will see edit controls that allow you to modify the data using the interface.

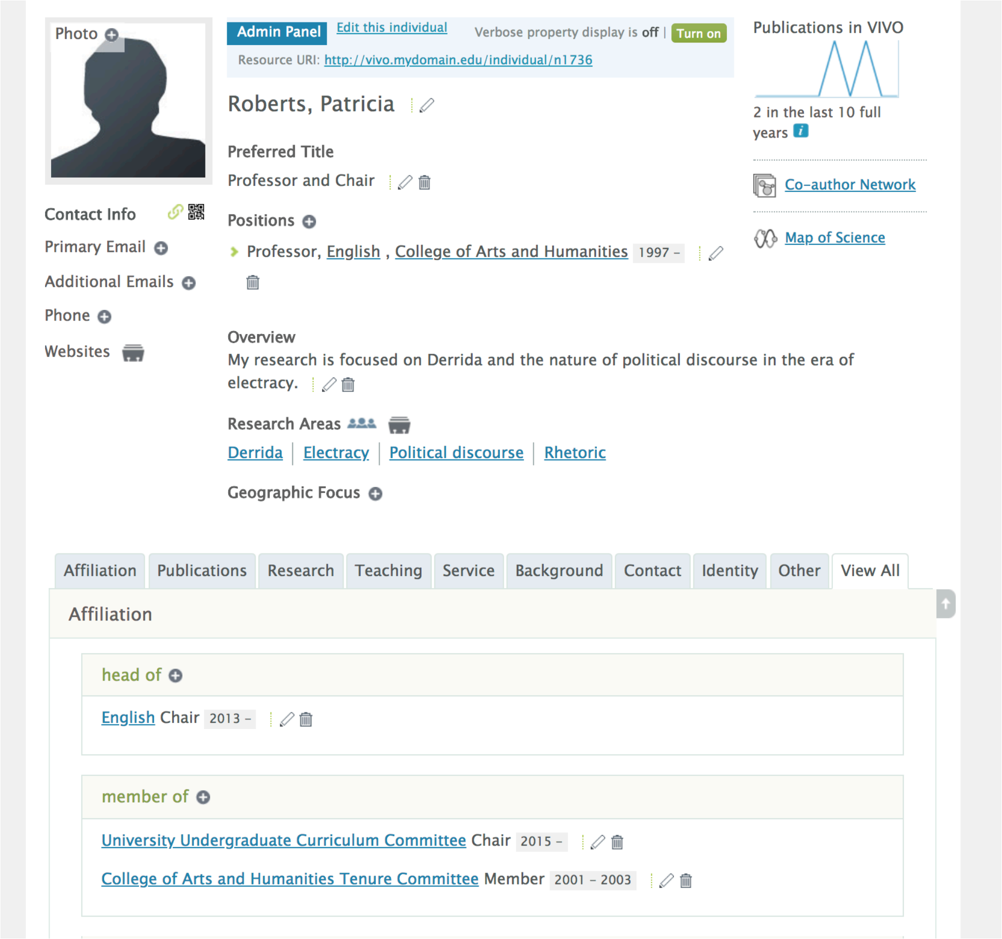


Photo
Admin Panel
Edit this individual
Turn on
Publications in VIVO
Co-author Network
Map of Science
Contact Info
Primary Email
Additional Emails
Phone
Websites
Affiliation
Publications
Research
Teaching
Service
Background
Contact
Identity
Other
View All

Verbose property display is off | Turn on

Resource URI: http://vivo.mydomain.edu/individual/n1736

Roberts, Patricia

Preferred Title
Professor and Chair

Positions
Professor, English , College of Arts and Humanities 1997 –

2 in the last 10 full years

Overview
My research is focused on Derrida and the nature of political discourse in the era of electracy.

Research Areas
Derrida | Electracy | Political discourse | Rhetoric

Geographic Focus

Affiliation

head of
English Chair 2013 –

member of
University Undergraduate Curriculum Committee Chair 2015 –
College of Arts and Humanities Tenure Committee Member 2001 – 2003

On the person's profile, click "Turn On" at the top of the page to turn on verbose property display. This shows you the properties VIVO is using to store each piece of information on the page, and attributes assciated with each of the properties.

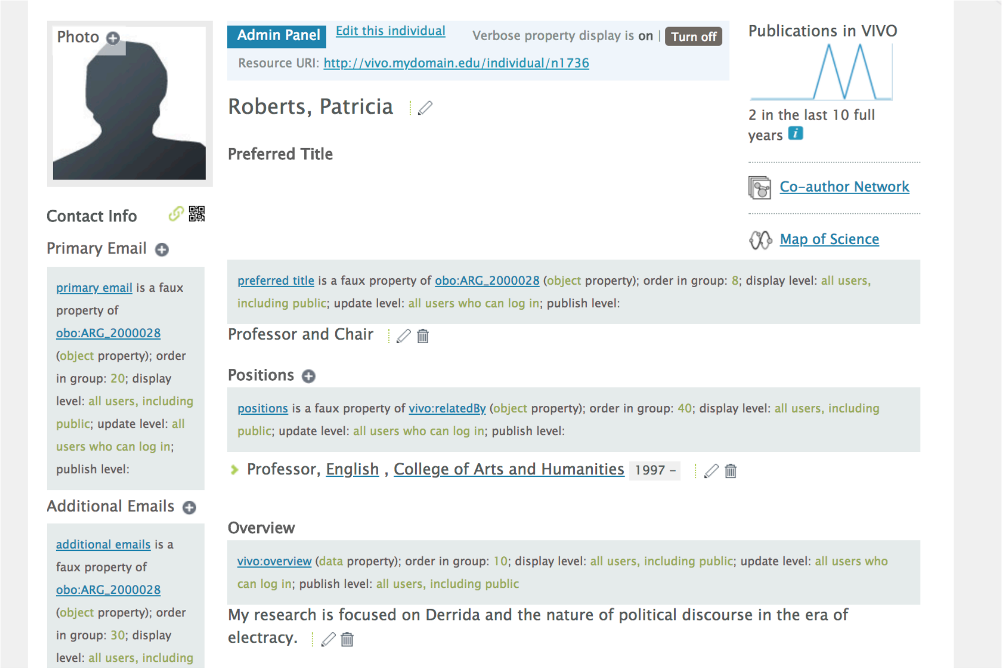


Photo
Admin Panel
Edit this individual
Verbose property display is on | Turn off
Publications in VIVO
2 in the last 10 full years
Co-author Network
Map of Science

Resource URI: http://vivo.mydomain.edu/individual/n1736

Roberts, Patricia

Preferred Title

preferred title is a faux property of obo:ARG_2000028 (object property); order in group: 8; display level: all users, including public; update level: all users who can log in; publish level:

Professor and Chair

Positions

positions is a faux property of vivo:relatedBy (object property); order in group: 40; display level: all users, including public; update level: all users who can log in; publish level:

Professor, English , College of Arts and Humanities 1997 –

Overview

vivo:overview (data property); order in group: 10; display level: all users, including public; update level: all users who can log in; publish level: all users, including public

My research is focused on Derrida and the nature of political discourse in the era of electracy.

Contact Info

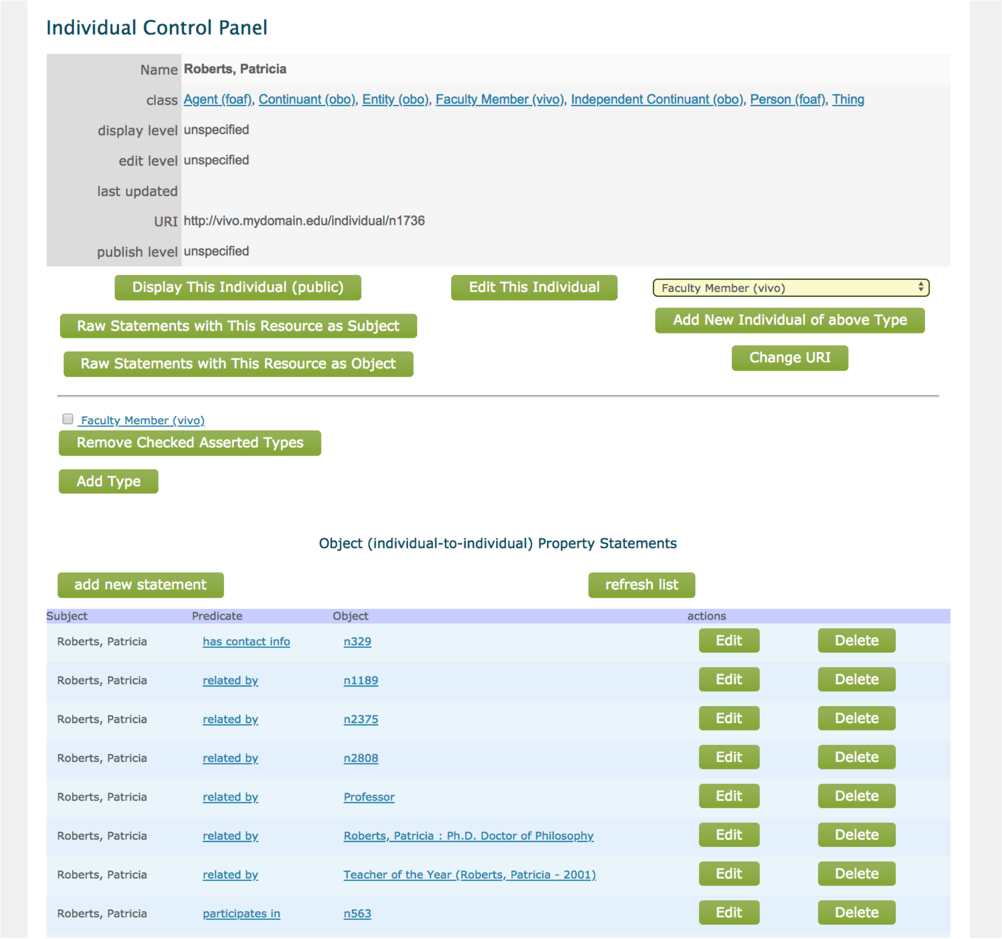
Primary Email

primary email is a faux property of obo:ARG_2000028 (object property); order in group: 20; display level: all users, including public; update level: all users who can log in; publish level:

Additional Emails

additional emails is a faux property of obo:ARG_2000028 (object property); order in group: 30; display level: all users, including

On a person's profile, click "Edit this individual". You will be presented with an internal view providing information about the data being used to create the profile.



Individual Control Panel

Name	**Roberts, Patricia**
class	Agent (foaf), Continuant (obo), Entity (obo), Faculty Member (vivo), Independent Continuant (obo), Person (foaf), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.mydomain.edu/individual/n1736
publish level	unspecified

Display This Individual (public)
Edit This Individual
Faculty Member (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
☐ Faculty Member (vivo)
Remove Checked Asserted Types
Add Type

Object (individual-to-individual) Property Statements

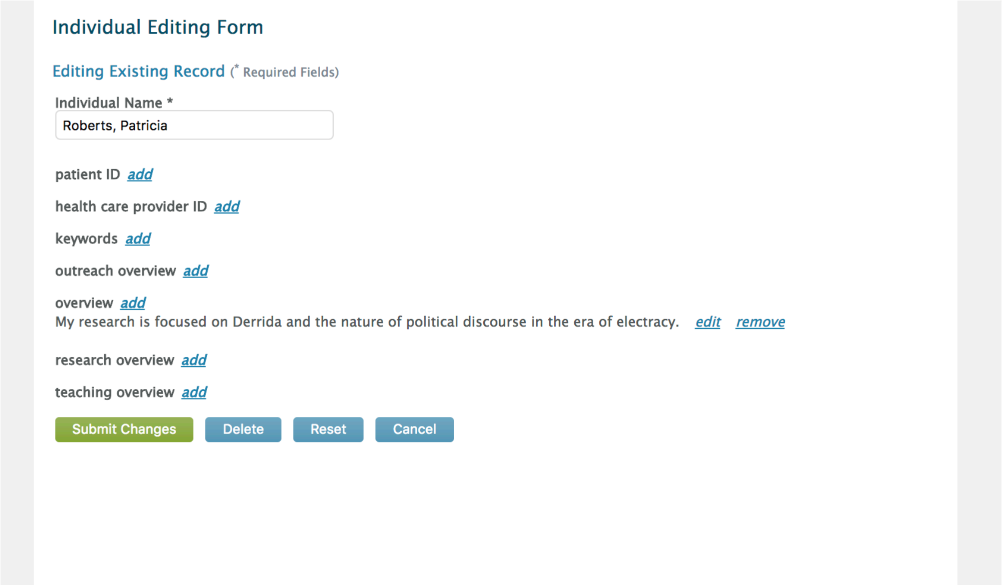
add new statement
refresh list

Subject	Predicate	Object	actions	
Roberts, Patricia	has contact info	n329	Edit	Delete
Roberts, Patricia	related by	n1189	Edit	Delete
Roberts, Patricia	related by	n2375	Edit	Delete
Roberts, Patricia	related by	n2808	Edit	Delete
Roberts, Patricia	related by	Professor	Edit	Delete
Roberts, Patricia	related by	Roberts, Patricia : Ph.D. Doctor of Philosophy	Edit	Delete
Roberts, Patricia	related by	Teacher of the Year (Roberts, Patricia - 2001)	Edit	Delete
Roberts, Patricia	participates in	n563	Edit	Delete

At the top of the display is a list of the datatype properties and their values to be edited.

Below that is a list of Object properties showing connections between the individual and other entities in VIVO. Studying these connections can help you understand how VIVO stores data in a connected graph.

Click on "Edit this Individual" on the Individual Control Panel.



Individual Editing Form

Editing Existing Record (* Required Fields)

Individual Name *
Roberts, Patricia

patient ID add

health care provider ID add

keywords add

outreach overview add

overview add
My research is focused on Derrida and the nature of political discourse in the era of electracy. edit remove

research overview add

teaching overview add

From this page you can edit or add values for any of the properties listed.

4.3.5 Resetting Your Database

When you are finished exploring the sample data, you will want to reset VIVO to use the database you used when installing VIVO. You recorded the name of this database when following the steps above in Preparing Your Database. To reset your database, follow the steps below:

1. Shutdown Tomcat
2. edit runtime.properties and change the name of the database from the name of the sample database to the name of the database you used to install VIVO.
3. Start Tomcat

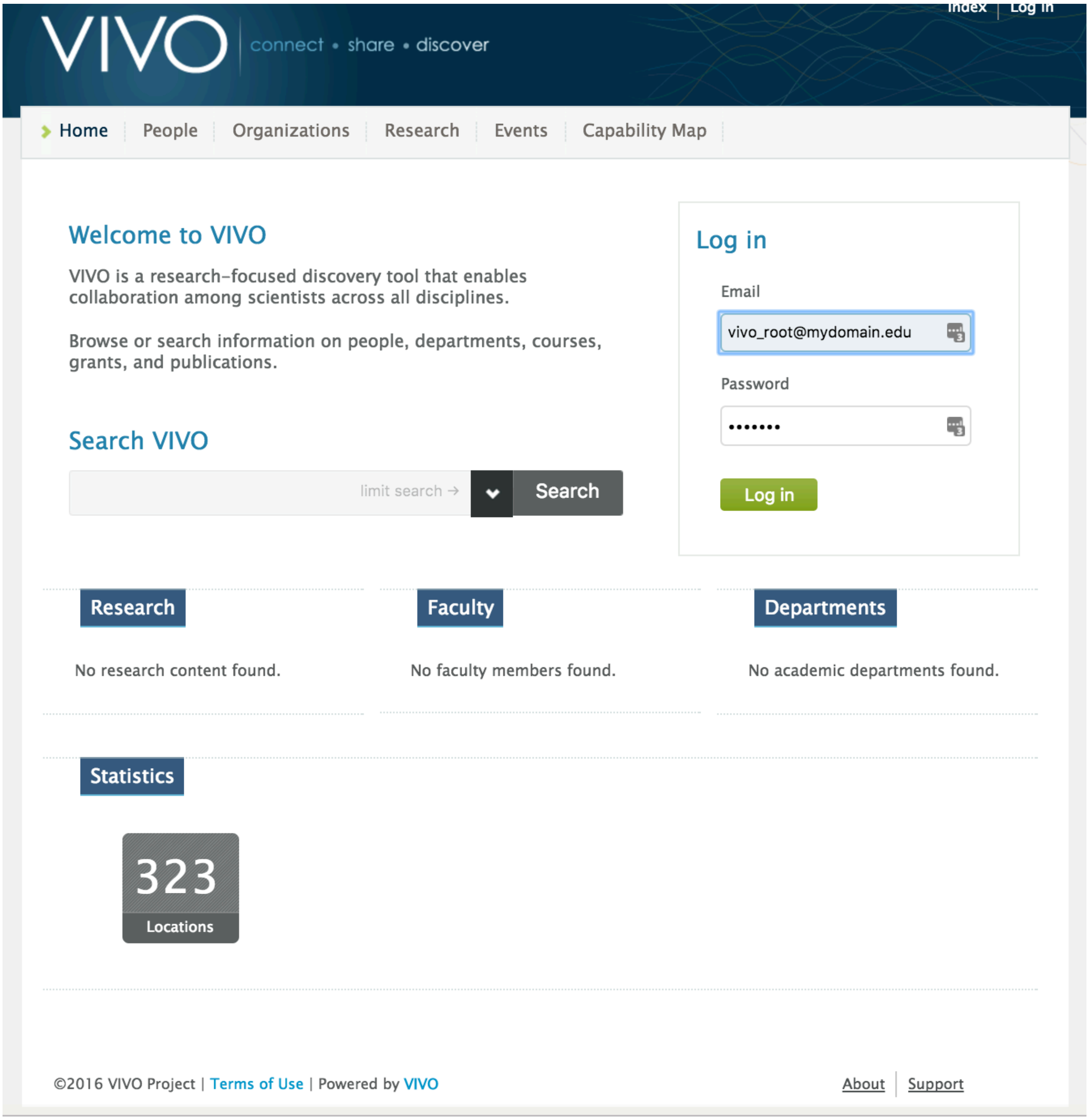
Now VIVO will be using the database you used when installing VIVO. If you would like to use the sample data again, just repeat these steps, naming the sample data database as the one you would like to use.

4.4 Restoring VIVO to First Time State

When Installing VIVO, a TDB database has been created in the subfolder of VIVO home directory defined by tdbContentTripleSource in the applicationSetup.n3 file (by default the subfolder is tdbContentModels). To restore VIVO to first time state, simply delete this subfolder.

Now restart Tomcat. VIVO will detect an empty data store, and proceed to load it from the `firsttime`, `everytime` and `filegraph` directories. Tomcat will start, but search indices may take additional time to complete. When reindexing is complete, refresh your browser to see your VIVO returned to first time state.

Your first time VIVO home should appear as shown below.


Index | Log in
VIVO
connect • share • discover
Home
People
Organizations
Research
Events
Capability Map

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

limit search →
Search

Log in

Email: vivo_root@mydomain.edu

Password: •••••••

Log in

Research

No research content found.

Faculty

No faculty members found.

Departments

No academic departments found.

Statistics

323 Locations

©2016 VIVO Project | Terms of Use | Powered by VIVO

About
Support

5 Preparing for Production

- Minimum Configuration (see page 63)
- About Page and Support Link (see page 64)
- Create, Assign, and Use an Institutional Internal Class (see page 65)
- Adding User Accounts (see page 68)
- Adding Site Information (see page 69)
- Simple Theming (see page 70)
- Configuring Web Analytics Tools (see page 73)
- Adding the Data Distribution API (see page 74)
- Configuring External Vocabularies (see page 74)

5.1 Overview

Once you have an initial VIVO running, loaded sample data, and explored the interface, you should be ready to make some choices about your production VIVO.

In this section, we will see the choices that need to be made before launching a production VIVO site. Choices will lead to restarting VIVO to confirm things are working as expected, and clearing out data that was created during testing.

5.2 Minimum Configuration

5.2.1 Email

VIVO should be configured to send email using an smtp service of your choice. VIVO sends email from its contact form. VIVO also sends email to users when changes are made to their accounts. Modify the two lines below in runtime.properties, then restart Tomcat. In the case the SMTP server is not using common port (25), or requires authentication, please uncomment the three lines at the end of the listing below and set up parameters for your server.

Email parameters in runtime.properties

```
#
# Email parameters which VIVO can use to send mail. If these are left empty,
# the "Contact Us" form will be disabled and users will not be notified of
# changes to their accounts.
#

email.smtpHost = smtp.mydomain.edu
email.replyTo = vivoAdmin@mydomain.edu
# email.port = 25
# email.username = username
# email.password = password
```

5.2.2 Namespace

The namespace parameter is the single most important parameter in your VIVO configuration. It is used in every triple created by VIVO. It should match the domain name of your VIVO production site. So, if your VIVO production site will be reached on the Internet with a web address of http://vivo.mydomain.edu the Vitro.defaultNamepsace parameter is runtime.properties should be set as shown below.

Namespace parameter in runtime.properties

```
  #
  # This namespace will be used when generating URIs for objects created in the
  # editor. In order to serve linked data, the default namespace must be composed
  # as follows (optional elements in parentheses):
  #
  # scheme + server_name (+ port) (+ servlet_context) + "/individual/"
  #
  # For example, Cornell's default namespace is:
  #
  # http://vivo.cornell.edu/individual/
  #

Vitro.defaultNamespace = http://vivo.mydomain.edu/individual/
```

5.2.3 Additional Configuration

For additional configuration parameters see Configuration Reference (see page 434)

5.3 About Page and Support Link

5.3.1 Overview

VIVO is delivered with an "About" page, describing VIVO at your institution, and intended to provide information to your users about your VIVO site. You may wish to include information about your data holdings and sources, your policies, and links to local documentation. The support link is a link to your support. It might be a mail link, a link to a contact form, or a link to web, wiki or other support information.

As delivered, the About page provides generic information about the project, and a place holder for your information. **You will want to change this.**

As delivered, the Support link is inoperative. **You will want to change this**

5.3.2 About Page

The About Page is constructed by the VIVO application. To change it, you will need to be in "firsttime" status – that is, your VIVO database must be empty. In "firsttime" status, the VIVO application looks to see if there is an About page to build, and builds it if necessary. Text for the About page can be found in an RDF file. Your will need to edit this file and restart VIVO with an empty database.

1. Navigate to any language aboutPage you would like to modify (the example is for en_US): < `vivo-home-dir>/rdf/i18n/en_US/display/firsttime/aboutPage.n3`
2. Edit `aboutPage.n3` This file contains properties used by the VIVO application to build the About page. You will edit the htmlValue of about:ABOUTDG. Replace the text following the triple quotes and up to the final triple quotes with the HTML you wish to display on your About page. Leave the remainder of the page untouched – the file contains directives to the VIVO application so that it can build a new About Page.
3. Restoring VIVO for the first time state - link (see page 61)
4. Visit your About page to check it

5.3.3 Support Link

To provide a support link:

1. Find the file `footer.ftl` for the theme you are using. To find the footer file, go to your VIVO source directory and use the command

```
find . -name '*footer*.ftl'
```

2. Edit `footer.ftl` to provide your support link. This file is a Freemaker template file. Leave the remainder of the file untouched – the file contains directives to the VIVO application to display a page footer on every page produced by VIVO.
3. Rebuild VIVO
4. Test your link

5.4 Create, Assign, and Use an Institutional Internal Class

5.4.1 Overview

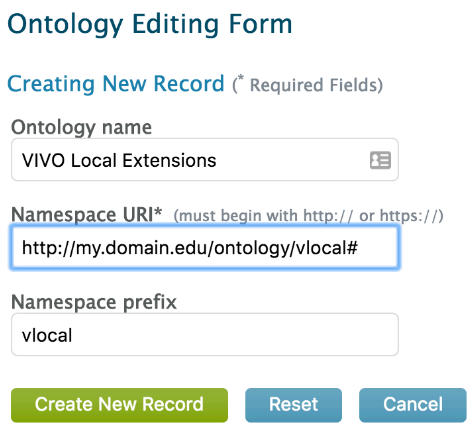
VIVO supports the concept of an Institutional Internal Class, a class that you can create and assign to your people, and other entities, to indicate that they are part of your institution. Using an Institutional Internal Class, you can limit VIVO's displays of entities to those in your institution.

5.4.2 Create an Institutional Internal Class

Create a local ontology if you do not already have one. If you have one, add a class to your existing local ontology.

Go to Site Admin > Ontology list > Add new ontology

Add your new ontology (see the example below)



Ontology Editing Form

Creating New Record (* Required Fields)

Ontology name: VIVO Local Extensions

Namespace URI* (must begin with http:// or https://): http://my.domain.edu/ontology/vlocal#

Namespace prefix: vlocal

Ontology name: Whatever you want. The name you give will appear in the list of VIVO ontologies.

Namespace: Must be your domain name as specified in your runtime.properties, followed by "/ontology/" followed by a name of your choice, followed by the '#' sign.

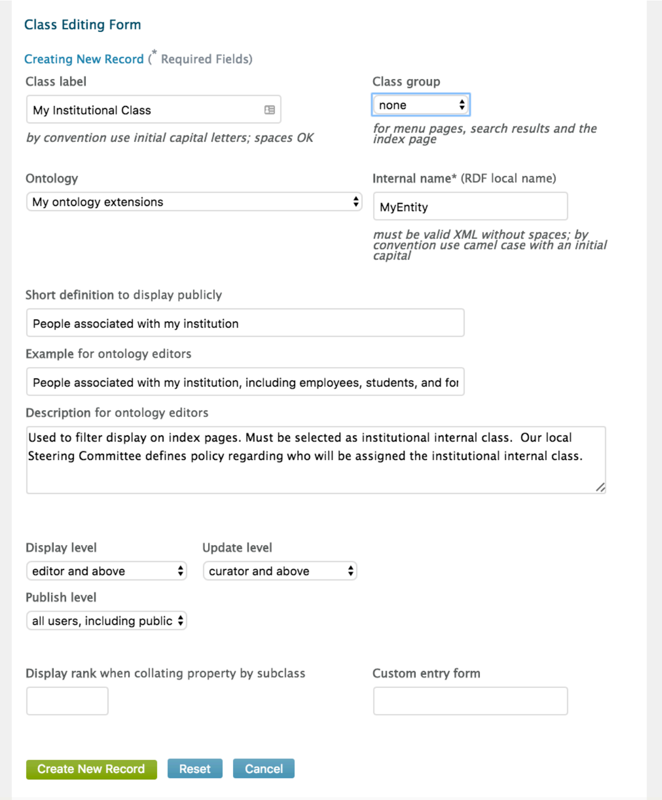
Namespace prefix: a short word. This word will appear in the prefix list in your SPARQL windows and will be used by you in any SPARQL queries referring to your local ontology.

Submit Changes

Add a new class to your local ontology

Go to 'Hierarchy of Classes Defined in This Namespace' > Add New Class

Add your new class (see the example below)



Class Editing Form

Creating New Record (* Required Fields)

Class label: My Institutional Class

by convention use initial capital letters; spaces OK

Class group: none

for menu pages, search results and the index page

Ontology: My ontology extensions

Internal name* (RDF local name): MyEntity

must be valid XML without spaces; by convention use camel case with an initial capital

Short definition to display publicly: People associated with my institution

Example for ontology editors: People associated with my institution, including employees, students, and fo

Description for ontology editors: Used to filter display on index pages. Must be selected as institutional internal class. Our local Steering Committee defines policy regarding who will be assigned the institutional internal class.

Display level: editor and above

Update level: curator and above

Publish level: all users, including public

Display rank when collating property by subclass:

Custom entry form:

Class label: Text, describes the class. This will be visible on the VIVO interface.

Class Group: People. This allows the class to be used to restrict the display of people to those people who have the institutional class you are defining.

Ontology: Select your previously created local ontology from the drop down menu

Internal Name: This word will be used in your SPARQL queries, along with the local ontology prefix to refer to your local class. In this example, the complete reference in SPARQL would be vlocal:MyEntity.

Short definition to display publically: Use language your users will understand

Example for ontology editors: More detail. Can be very technical.

Description for ontology editors: Will appear in the ontology editor. Remind future ontology editors of the purpose of the class and how one will know what entities should be in the class.

Display level: Set to editor and above from the drop down

Update level: Set to curator and Above from the drop down

Publish level: All users including public

Once you are satisfied with the values, press Create New Record

5.4.3 Assign your Institutional Internal Class

Method 1: (Manual)

Go to the person in the UI

Click Edit Individual

Click Add Type

Select your Institutional Internal Class from the drop down

Method 2: (Bulk)

Create a set of RDF, one triple per person you would like to have in the institutional class. Each triple will look like

```
<personuri>  rdf:type vlocal:MyEntity .
```

Go to Site Admin -> Add/remove RDF Data and add your triples

5.4.4 Use your Institutional Internal Class

Define institutional internal class

Go to Site Admin > Institutional internal class

Select your new class from the dropdown menu

To restrict display to only those people in your institution,

Go to Site Admin > Page Management > People

Click the plus sign to expand the ‘Browse Class Group’ box

Check ‘Only display people within my institution'

Click "Save this Content'

5.5 Adding User Accounts

The root user and users with the role Site Admin are allowed to create user accounts. To create a user, one has to follow these steps:

1. Enter the 'Site Administration'.
2. Choose 'User Accounts' from the menu 'Site Configuration'.
3. Click on 'Add new account'
4. Mandatory fields to enter are 'Email Address', 'First name', and 'Last name'.
5. Choose a role for the respective user. More information about the role management can be found in the 'System Administration' section of this documentation on the page 'Creating and Managing User Accounts (see page 294)'.
6. (optional) To support self-edit, add a value for the Matching ID. The Matching ID is a datatype property set in `runtime.properties` . The value you enter on the user account must match the value on the user's profile.
7. Click 'Add new account'

In normal operation, an email will be sent to the address entered notifying that an account has been created. It will include instructions for activating the account and creating a password.

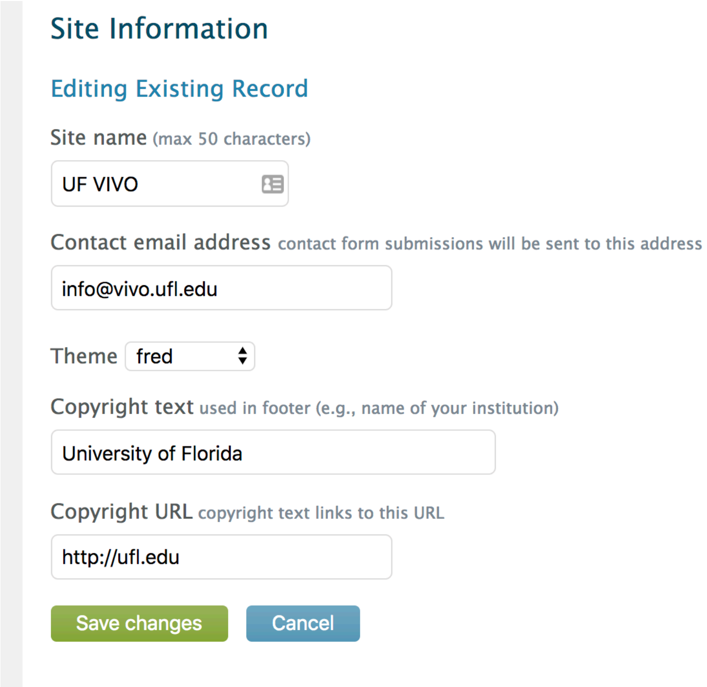
5.6 Adding Site Information

Using Site Admin → Site Information, you can:

1. Set the site name (typically something like Scholars@Cornell or GWU Experts, or UF VIVO). The site name is used on several pages when explaining the site to users.
2. Contact Email address. provide an email address that will receive contact info from VIVO. Typically this is set to a generic email address or list that can be answered by the people responsible for user support. Example: info@vivo.myschool.edu[62]
3. Set the theme. See below. You can customize the theme - link (see page 70)
4. Copyright holder. This is typically the name of your institution. Example: George Washington University
5. Copyright URL. A URL that will be visited when a user clicks on the copyright link at the bottom of each VIVO page.

Site information is stored in the VIVO configuration triple store.

An example is shown below:



Site Information

Editing Existing Record

Site name (max 50 characters)

UF VIVO

Contact email address contact form submissions will be sent to this address

info@vivo.ufl.edu

Theme fred

Copyright text used in footer (e.g., name of your institution)

University of Florida

Copyright URL copyright text links to this URL

http://ufl.edu

---

[62] mailto:info@vivo.myschool.edu

5.7 Simple Theming

5.7.1 Overview

VIVO uses Freemarker templates to create its web pages. These templates can be modified to alter VIVO's appearance. Most VIVO sites alter VIVO's default appearance. VIVO has a "Site Info" capability that allows a few presentation elements to be modified from the Site Admin menu. Modifying other elements involves creating a theme and modifying that theme. Here we describe simple modifications that most sites want to make before they go into production. For a detailed description of theming and templates, see Creating a custom theme (see page 210)

Customization

The method described below is a customization of VIVO. This customization **will be** lost in an upgrade, or code refresh unless you preserve it and reapply the customization after an upgrade. A method for preserving customizations during an upgrade can be found here: Preserving Customizations During Build (see page 51). It is best to plan ahead to preserve customizations.

5.7.2 Create a Theme

VIVO comes with a standard theme, called `wilma.` `wilma` is in the folder in `vivo/installer/webapp/target/vivo/themes`.

To create a new theme, choose a name for your new theme. In these examples below we will call the new theme `fred`.

Copy the `wilma` directory and its contents to a new directory called `fred`. `fred` must also be in `vivo/installer/webapp/target/vivo/themes.`

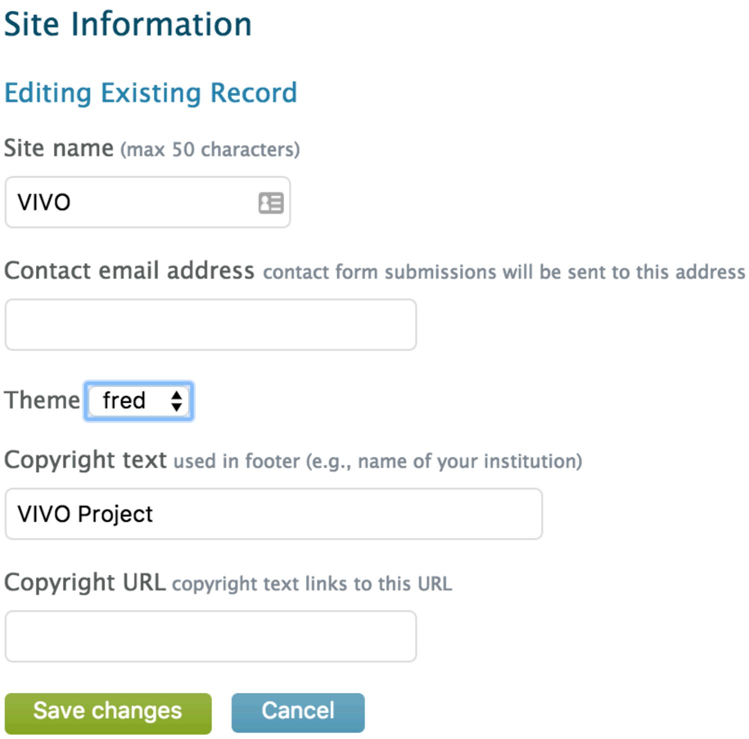
Your new theme will contain CSS files, image files, and Freemarker[63] templates.

Copy the `vivo/installer/home/target/vivo/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_wilma.ttl into vivo/installer/home/target/vivo/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_fred.ttl`, and substitute (find and replace) the term `wilma` with the term `fred` inside the new file. The previous example is working for English (en_US) user interface, the same action should be done for other languages used in the VIVO instance.

Run the Maven install to deploy your new theme to the Tomcat container. Restart the VIVO Tomcat process. You can then go to the ***Site Admin*** page and choose ***Site Information***, to select your theme as the current one.

---

[63] http://freemarker.org/



Site Information

Editing Existing Record

Site name (max 50 characters)

VIVO

Contact email address contact form submissions will be sent to this address

Theme fred

Copyright text used in footer (e.g., name of your institution)

VIVO Project

Copyright URL copyright text links to this URL

5.7.3 Modify Header with Logo

Now that you have a theme, we can modify the templates in the theme to change VIVO's look. Let's start with the header. You can replace VIVO's default logo ("VIVO: connect * share * discover") with your organization's logo. VIVO will look best if your logo has a height of 59px.

1. Add your logo to fred/images
2. Edit fred/css/wilma.css to replace the VIVO logo with your logo. Find h1.vivo-logo and change the URL of the background image to the name of your logo in images.
3. Build VIVO with Maven and restart Tomcat.
4. Check to see that your logo is in place

5.7.4 Modify Footer

The footer for VIVO can be found in fred/templates/footer.ftl. You may want to change it's look, or merely change where VIVO sends people for Support (by default, VIVO send people to the VIVO web site – you may have a preferred location, or mail address for support.

1. Edit fred/templates/footer.ftl to specify a URL for Support. Find menu_contactus and change the href in the anchor tag to point at the desired Support location – this could be a mailto:emailaddress or a URL for your a support web site.

2. Build VIVO with Maven and restart Tomcat
3. Check to see that your support link is in place

5.7.5 Add Your Colors

Most sites have one or more colors that are used throughout their site and are identified with their organization as part of their branding program. You may want to check with your institution regarding their branding and identify two colors by RGB code that should be used in your VIVO. One color will be used for headings and should be dark enough for use as a text color on a white background. A second color will be used for a background color to replace VIVO's blue.

1. Identify your headings color and your background color
2. Edit fred/css/wilma.css
3. Replace h2, h3, h4, h5, h6 colors with your headings color. Find each of the corresponding color assertions for these headings and change them to the color you prefer for headings.
   a. You may also wish to change the colors for "Index" and "Login" at the top of the page. The default is white, which works well on a dark background. But if your background color is light (see below), you may want to make these the same as your headings color. The style elements are named ul#header-nav
   b. Similarly, you may wish to change the colors for "Terms of Service" and such as the bottom of the page, to make them the same as your headings color. The elements are named ul#footer-nav
4. Replace the background color with your background color. VIVO uses a background color and a background image to create the wavy lines in its header. To replace the background color and the wavy lines with your background color, find and replace the background in the body tag, with

   ```
   background: #xxxxxx center 0 no-repeat;
   ```

   where `#xxxxxx` is the RGB hex code for your background color
5. Build VIVO with Maven and restart Tomcat
6. Check to see that your colors are in place

5.7.6 Additional Theming

There is no end to what can be done with theming and VIVO's templates. For examples of sites that have customized their VIVO with a local theme, and for more information on theming, see Creating a custom theme (see page 210)

5.7.7 Experimental Theming

New in VIVO 1.10 is the beta tenderfoot theme. This beta theme is based on Bootstrap and provides a responsive experience on devices of various screen sizes. To try tenderfoot, select tenderfoot in Site Admin → Site Information. To create a custom version of tenderfoot, follow the instructions above for creating a theme, starting with tenderfoot rather than wilma. Note that tenderfoot is a beta theme and may not have all the features needed for production.

5.8 Configuring Web Analytics Tools

5.8.1 Background

There are various web analytics tools that allow insights into the behaviour of visitors to a website. These services can be useful in the case of VIVO to analyse and subsequently improve the use of the site overall or the uptake of specific VIVO features. There are several popular services. Among the most widely used are Google Analytics and Matomo (formerly Piwik).

5.8.2 Example: Google Analytics

Google Analytics is a popular, and proprietary analytics platform for measuring web traffic to a site. VIVO supports Google Analytics through a Freemarker Template dedicated for the purpose. You will need to sign up for Google Analytics, and use your site domain and your key as part of the configuration.

Google Analytics is **not required**. If you do not wish to implement Google Analytics, please skip this section.

Customization

The method described below is a customization of VIVO. This customization **will be** lost in an upgrade, or code refresh unless you preserve it and reapply the customization after an upgrade. A method for preserving customizations during an upgrade can be found here: Preserving Customizations During Build (see page 51). It is best to plan ahead to preserve customizations.

5.8.2.1 `googleAnalytics.ftl`

To implement Google Analytics for VIVO, follow the steps below.

1. Sign up for Google Analytics. Have your page key, your roll-up key and the site domain name of your VIVO production site ready.
2. If you are not using the default VIVO theme, copy `googleAnalytics.ftl` from the default theme (wilma) found here: `./webapp/src/main/webapp/themes/wilma/templates/googleAnalytics.ftl`
3. Edit `googleAnalytics.ftl` in the theme you are using. Provide your domain name, your page tracker key, and your roll-up tracker key.
4. `mvn install -Dskiptests`
5. Restart Tomcat. As you develop more experience with customizing the VIVO interface, and working with VIVO, you may be able to bypass some caching and make changes that will be picked up

immediately by Tomcat. In such cases you may not need to restart Tomcat. See Extending and Localizing VIVO (see page 128)

5.8.3 Other analytics platforms

It is possible to use other analytics platforms. For example, if you want to use Matomo, you can substitute the tracking script from Google Analytics with the one provided by Matomo.

5.9 Adding the Data Distribution API

5.9.1 Overview

The Data Distribution API (see page 339) is a powerful new tool for creating new APIs for VIVO. Using the Data Distribution API you can configure API endpoints that will respond with data from the VIVO content triple store in formats ready for use in other software.

The Data Distribution API is not included with VIVO 1.10 by default. Inclusion of the Data Distribution API is optional. If you would like to include the Data Distribution API in your VIVO, follow the process below.

5.9.2 Process for Adding the Data Distribution API

For the VIVO 1.10.x see https://github.com/vivo-community/vivo-data-distribution-api/blob/468526eeddf94032e299a83135d992247ca53233/site_builder/src/site/markdown/install_vivo_1_10.md

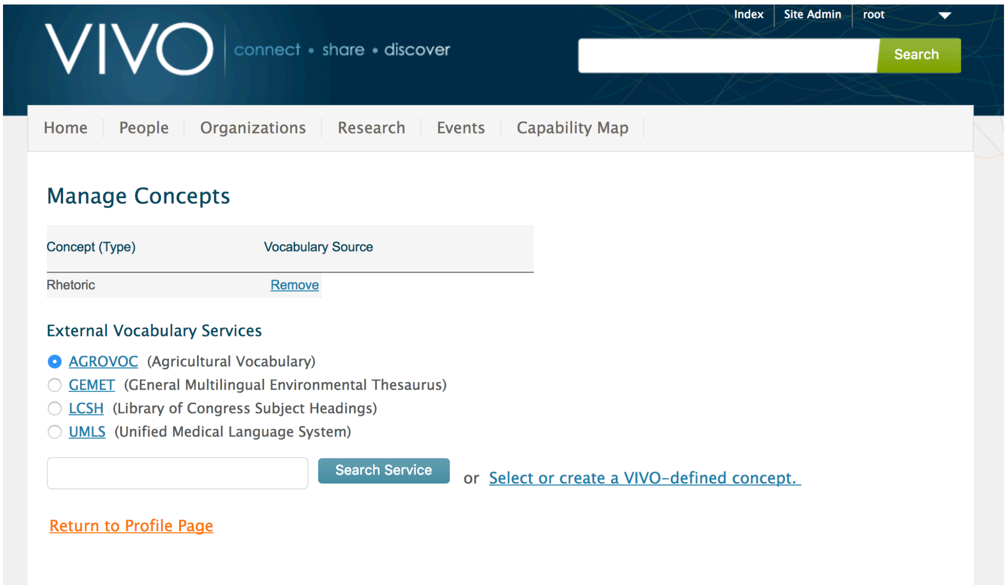
For the VIVO 1.11.x see https://github.com/vivo-community/vivo-data-distribution-api/blob/468526eeddf94032e299a83135d992247ca53233/site_builder/src/site/markdown/install_vivo_1_11.md

5.10 Configuring External Vocabularies

5.10.1 Overview

VIVO provides the ability to use external vocabularies to represent the research areas of scholars, and the concepts pertaining to scholarly works. External vocabularies that provide RDF can be used with VIVO. Using an external service, a curator, or page owner may query the external vocabulary for terms, and select terms representing the work or scholar. The terms are fetched from the external service and added to the VIVO triple store. Links from the work or person are added to connect the person or work to the selected term or terms.

See below



Manage Concepts

Concept (Type)	Vocabulary Source
Rhetoric	Remove

External Vocabulary Services

● AGROVOC (Agricultural Vocabulary)
○ GEMET (GEneral Multilingual Environmental Thesaurus)
○ LCSH (Library of Congress Subject Headings)
○ UMLS (Unified Medical Language System)

5.10.2 Configuring the UMLS External Vocabulary Service

The UMLS External Vocabulary Service is provided by the National Institutes of Health in the United States. They require an apikey to use the service. You can get an apikey at no charge from the NIH. Visit https://www.nlm.nih.gov/databases/umls.html for general information, and https://documentation.uts.nlm.nih.gov/rest/authentication.html for information on generating an API key.

Once you have the apikey, create a text file called umls.properties. Your file should look like the one below, with your value for apikey substituted.

umls.properties

```
apikey = yourumlsapikey
```

Place this file in the directory VIVO/installer/webapp/src/main/webResources/WEB-INF/classes/ in your source tree and reinstall VIVO. (You can copy the file directly to WEB-INF/classes inside the VIVO directory under Tomcat's webapps directory if you want to use UMLS right away without reinstalling VIVO and restarting Tomcat. Note that the file will not be preserved through future reinstallations unless you also add it to your source tree.)

5.10.3 Notes

When external concepts are added to VIVO, they retain their original URI from the external vocabulary. Since we have no way of knowing whether these URIs represent OWL classes or RDF instance data, VIVO does not assert a type for the concepts, which will therefore only be interpreted as being of type owl:Thing.

To add a new external vocabulary, see Adding External Vocabularies (see page 292)

6 Using VIVO

- Navigating VIVO (see page 76)
- Editing Your Profile (see page 77)
- Using Search (see page 84)
- Using Visualizations (see page 87)
- VIVO for Data Analysts (see page 95)

VIVO [Pronunciation: /viːvəʊ/ or *vee-voh*] is member-supported, open source software, and an ontology for representing scholarship. VIVO supports recording, editing, searching, browsing and visualizing scholarly activity. VIVO encourages research discovery, expert finding, network analysis and assessment of research impact. VIVO is easily extended to support additional domains of scholarly activity.

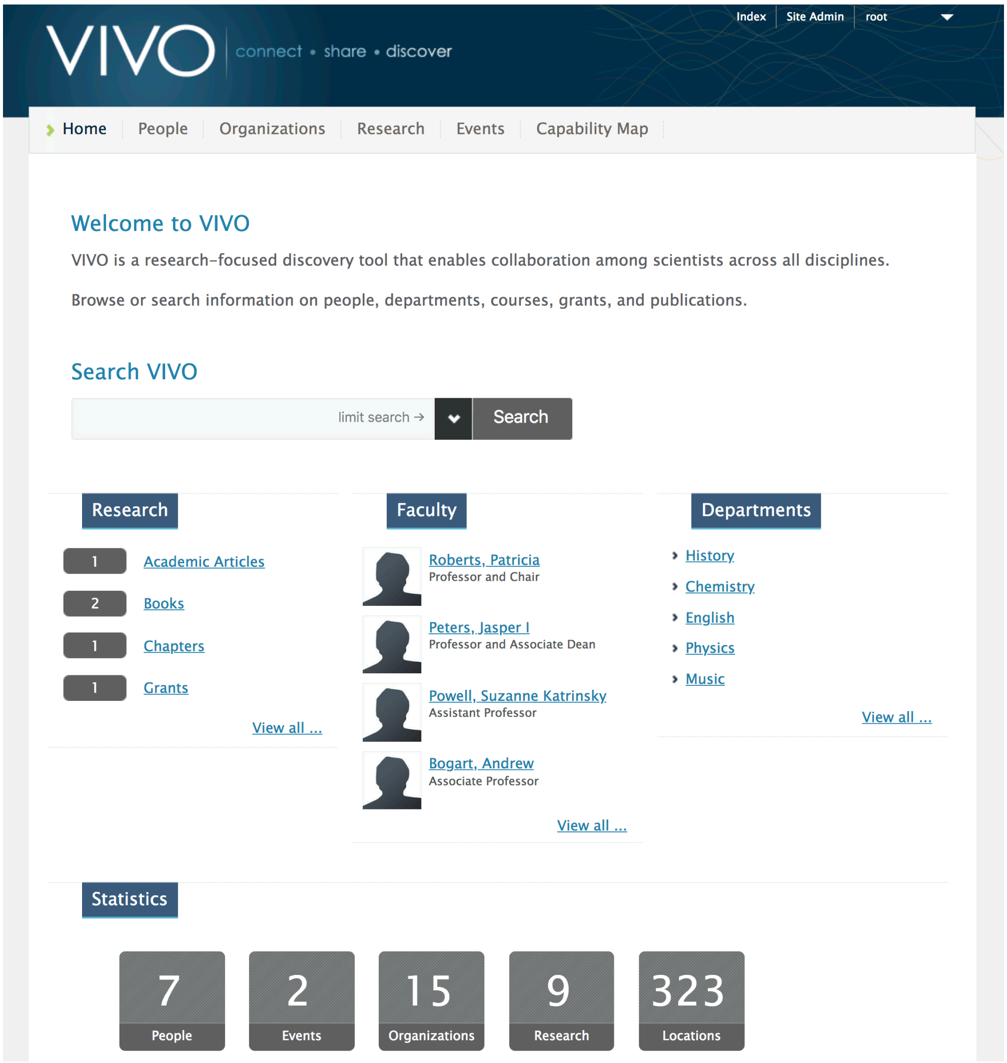
Here we will describe the basic features of VIVO and how you can use them. Many VIVO sites customize VIVO to add local features, enriching the description of the scholarship of their institution. Here we describe only the common features of VIVO. You may wish to ask your local VIVO providers for documentation describing the VIVO at your institution.

The examples to follow use an uncustomized VIVO. Your VIVO may look different.

6.1 Navigating VIVO

VIVO is navigated and browsed using a primary menu located along the upper portion of the website. By default, VIVO has five items in the primary menu; Home, People, Organizations, Research, and Events. Your VIVO may have been configured with additional menu items.

Beginning at the Home menu, notice the contents of each menu. Each menu contains a list of VIVO Individuals associated to particular "classes". Each menu shows the count of individuals in parentheses next to the superclass and subclass names. In addition, the "Index" link on the upper right corner displays a list of “VIVO individuals” by class and the associated count in parentheses.



Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

Research

1	Academic Articles
2	Books
1	Chapters
1	Grants

View all ...

Faculty

Roberts, Patricia	Professor and Chair
Peters, Jasper I	Professor and Associate Dean
Powell, Suzanne Katrinsky	Assistant Professor
Bogart, Andrew	Associate Professor

View all ...

Departments

- History
- Chemistry
- English
- Physics
- Music

View all ...

Statistics

7	2	15	9	323
People	Events	Organizations	Research	Locations

6.2 Editing Your Profile

Your site may be different

VIVO is easy to modify. Many VIVO sites have made changes to the way VIVO looks and the way VIVO operates. Some sites provide all data for their people and do not support editing of profiles. Others support requests for changes. Still others support login using institutional usernames and passwords and full editing of profiles. You will want to check with your local VIVO site maintainers regarding your site's editing practices. Here we assume that you can log on to your VIVO and edit your profile.

6.2.1 Overview

Your institution may supply a profile with your name, contact information, department affiliations, as well as current and past positions. In addition, publications which you authored, and grants on which you served as an investigator may also be automatically provided as part of your profile, and automatically updated by the institution.

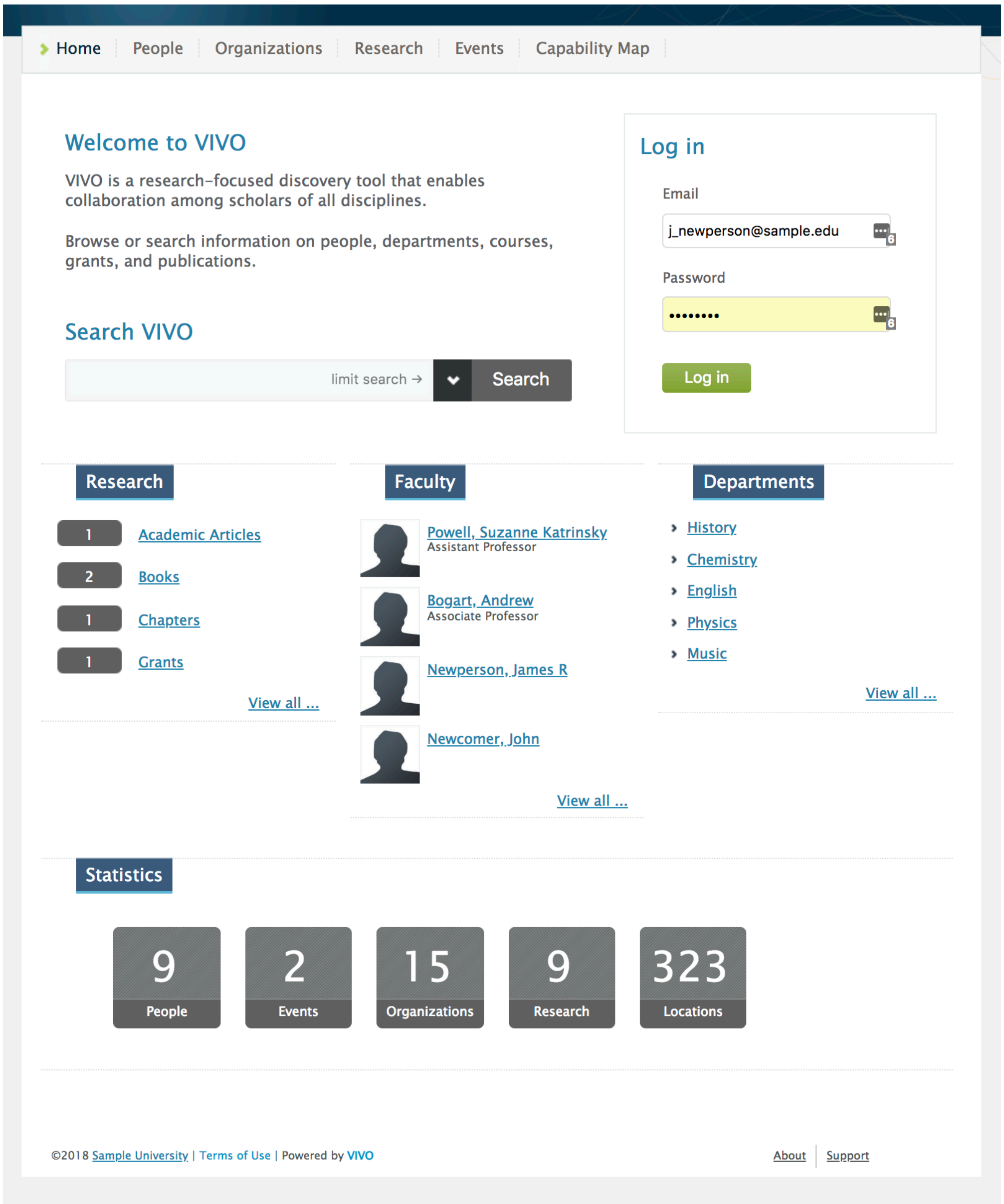
You will typically supply a photo, and a short text description, an overview, of your work.

Warning

All information on your VIVO profile should be considered public. The information will be displayed on the Internet, and will easily be found by those searching the Internet.

6.2.2 Sign on to VIVO

Go to your VIVO, enter your username (Email) and password, and click "Log in"



Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scholars of all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

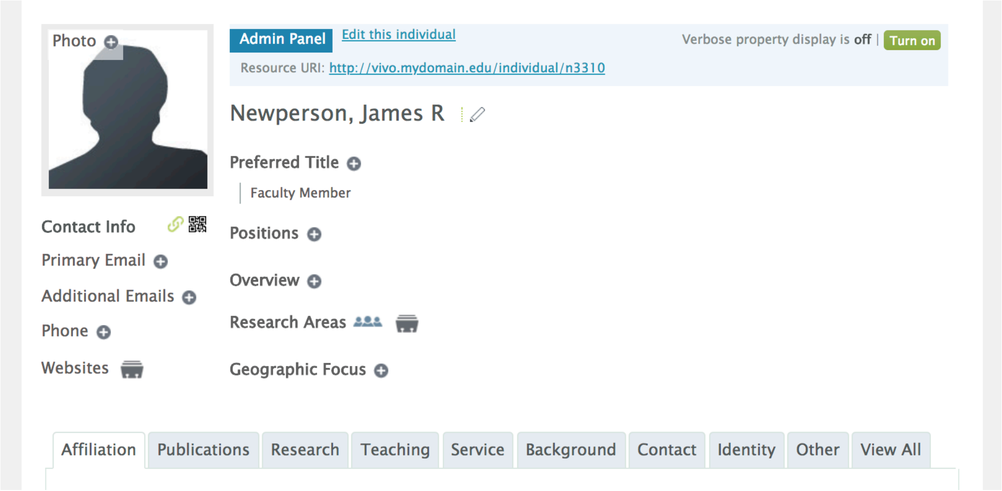
Log in

Email: j_newperson@sample.edu

Password: ••••••••

Home
People
Organizations
Research
Events
Capability Map
Search VIVO
limit search →
Search
Log in
Research
1 Academic Articles
2 Books
1 Chapters
1 Grants
View all ...
Faculty
Powell, Suzanne Katrinsky
Assistant Professor
Bogart, Andrew
Associate Professor
Newperson, James R
Newcomer, John
View all ...
Departments
History
Chemistry
English
Physics
Music
View all ...
Statistics
9 People
2 Events
15 Organizations
9 Research
323 Locations
©2018 Sample University | Terms of Use | Powered by VIVO
About
Support

VIVO takes you to your profile, ready to edit. VIVO presents editing controls that can be used to add, change, and delete items from your profile. Some sites will "lock down" some items on the profile and supply them for you. Let's see how to add a photo and an overview to your profile.



Resource URI: http://vivo.mydomain.edu/individual/n3310

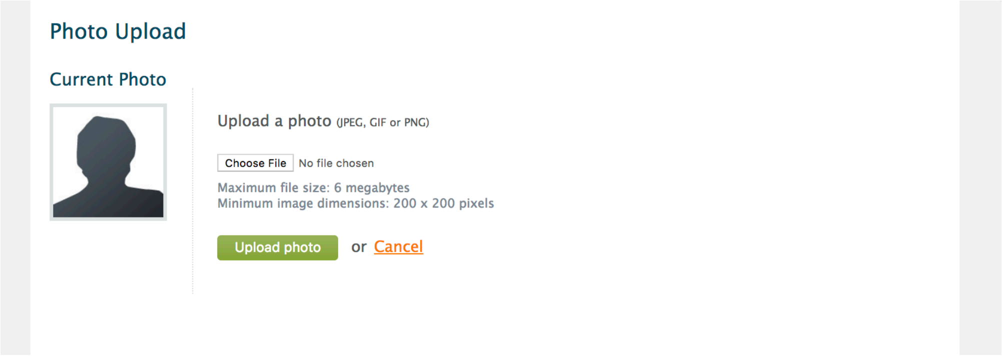
Newperson, James R

Preferred Title
Faculty Member

Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on
Contact Info
Primary Email
Additional Emails
Phone
Websites
Positions
Overview
Research Areas
Geographic Focus
Affiliation
Publications
Research
Teaching
Service
Background
Contact
Identity
Other
View All

6.2.3 Adding Your Photo

To add your photo, click on the plus sign ("+") on the photo in the upper left of the profile.



Upload a photo (JPEG, GIF or PNG)

No file chosen

Maximum file size: 6 megabytes
Minimum image dimensions: 200 x 200 pixels

Photo Upload
Current Photo
Choose File
Upload photo
or Cancel

Now choose your photo from your computer using the "Choose file" button. You will be able to navigate your computer's file system and select a photo. Once selected, the name of your photo will appear next to the Choose File button. Press "Upload photo"

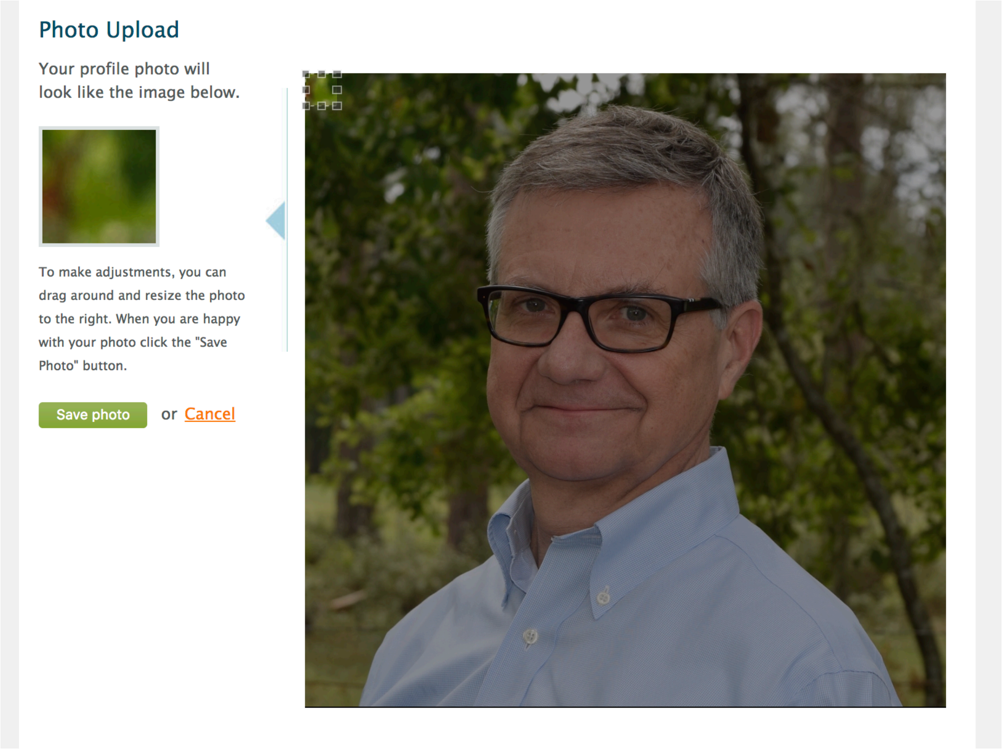


Photo Upload

Your profile photo will look like the image below.

To make adjustments, you can drag around and resize the photo to the right. When you are happy with your photo click the "Save Photo" button.

The image in the upper left shows how your photo will appear in VIVO. The image on the right shows the full image you uploaded. Note the cropping tool in the upper left of the right-hand image. Use it to select the portion of your photo you would like for your profile.

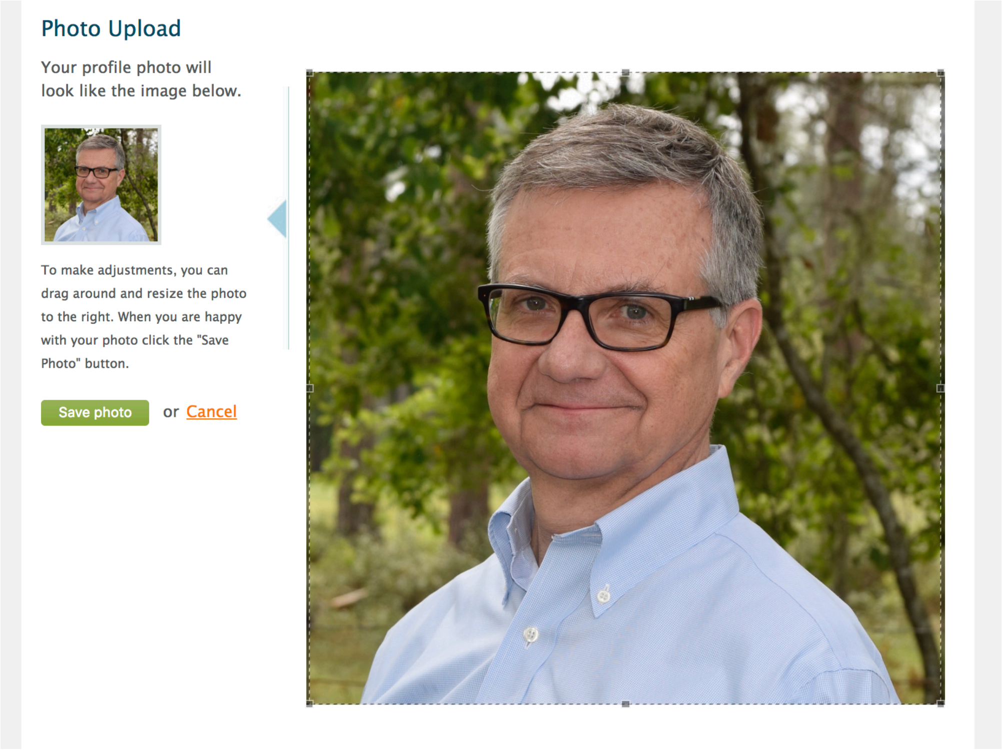
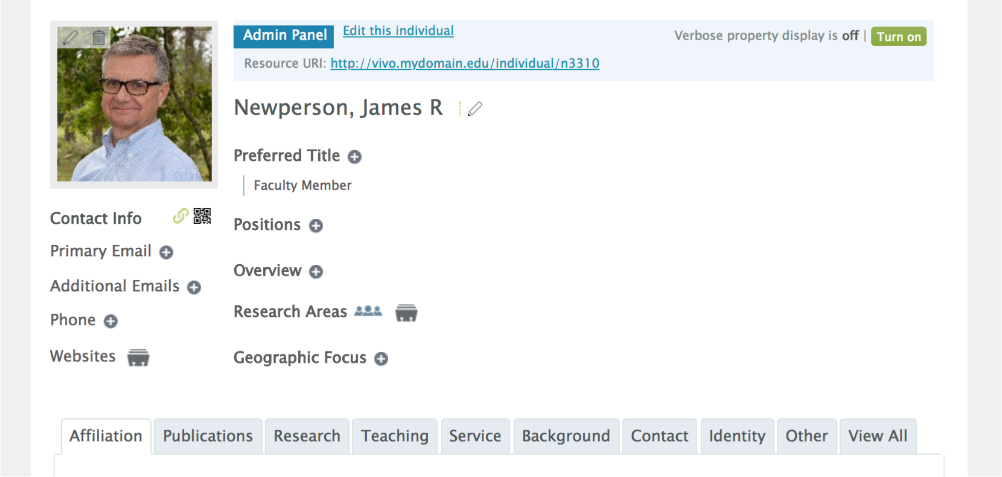


Photo Upload

Your profile photo will look like the image below.

To make adjustments, you can drag around and resize the photo to the right. When you are happy with your photo click the "Save Photo" button.

The best images for VIVO are square. Here we have selected just about the whole photo. The image in the upper left looks good. Now you can press "Save photo". VIVO returns you to your profile with the image you selected.

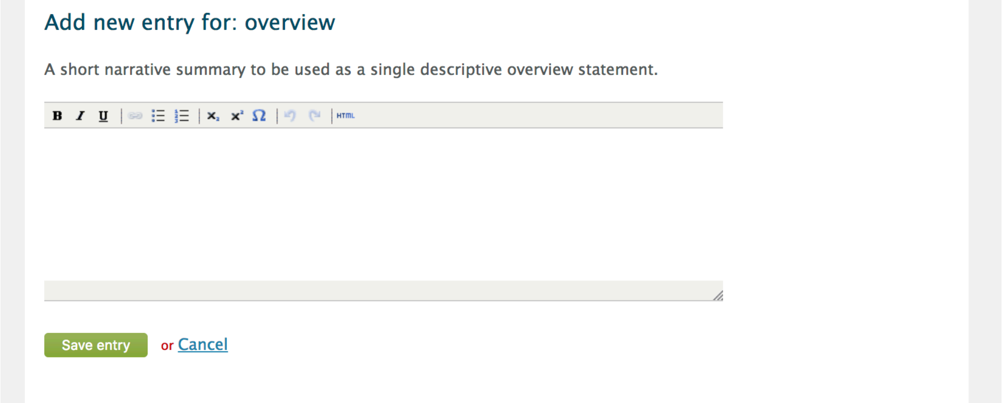

Admin Panel
Edit this individual
Verbose property display is off | Turn on
Resource URI: http://vivo.mydomain.edu/individual/n3310
Newperson, James R
Preferred Title
Faculty Member
Positions
Overview
Research Areas
Geographic Focus
Contact Info
Primary Email
Additional Emails
Phone
Websites
Affiliation
Publications
Research
Teaching
Service
Background
Contact
Identity
Other
View All

6.2.4 Adding an Overview

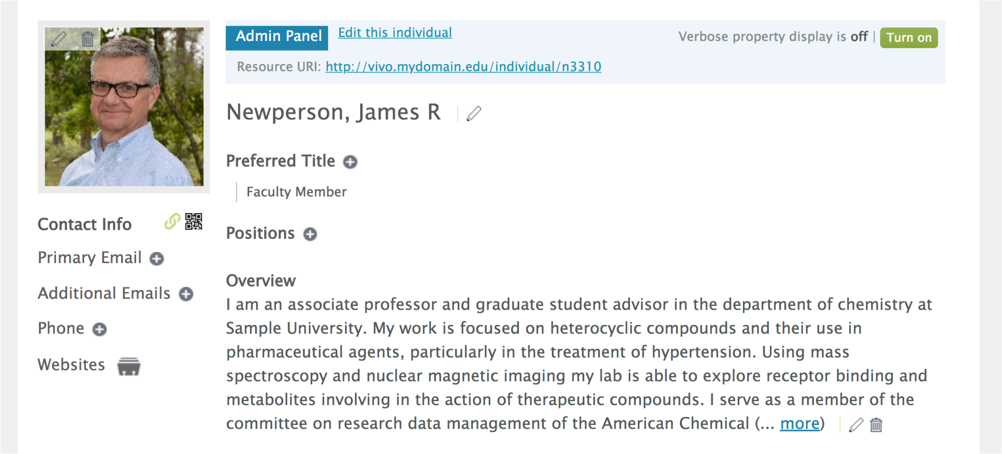
The Overview is an important part of your profile. You use the overview to describe your work to others. People using VIVO to find collaborators and expertise will search VIVO. VIVO searches your Overview. Search engines search the text in your Overview. A good overview is short (typically 50-100 words) and describes your interests, activities, and accomplishments.

You can edit your profile in VIVO at any time to keep it current with your interests.

To add your overview, click on the plus sign ("+") next to the word overview.


Add new entry for: overview
A short narrative summary to be used as a single descriptive overview statement.
Save entry
or Cancel

VIVO provides an HTML editor so that you can add highlights and links to your profile. When you are finished entering your overview, press "Save entry." You are returned to your profile with your overview on your profile.



Resource URI: http://vivo.mydomain.edu/individual/n3310

Newperson, James R

Preferred Title

Faculty Member

Contact Info

Primary Email

Additional Emails

Phone

Websites

Positions

Overview

I am an associate professor and graduate student advisor in the department of chemistry at Sample University. My work is focused on heterocyclic compounds and their use in pharmaceutical agents, particularly in the treatment of hypertension. Using mass spectroscopy and nuclear magnetic imaging my lab is able to explore receptor binding and metabolites involving in the action of therapeutic compounds. I serve as a member of the committee on research data management of the American Chemical (... more)

6.2.5 Positions and Publications

Most VIVO sites provide position information from institutional systems, including department affiliations, titles, and years of service. You will want to check with your site regarding how this is done for you.

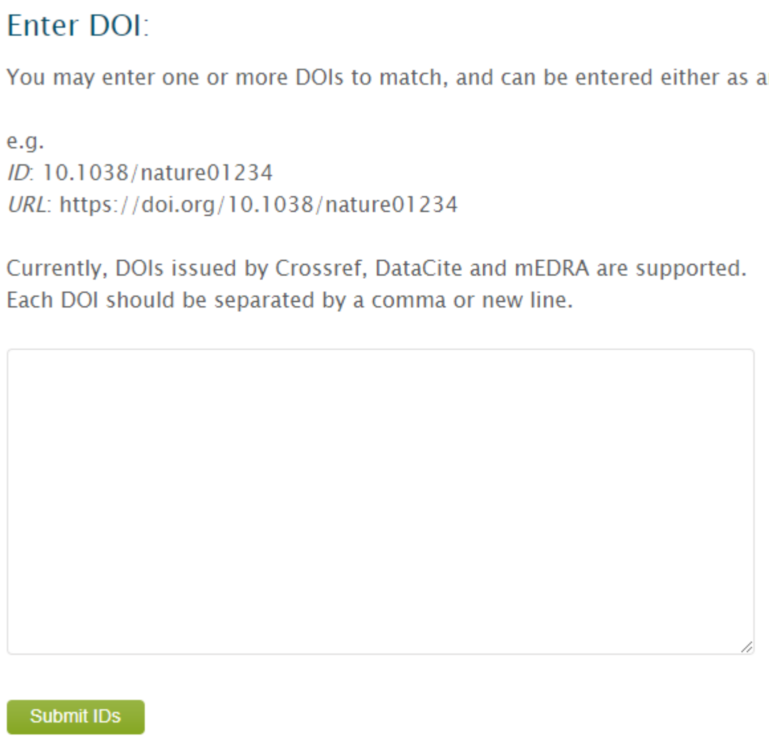
Most VIVO sites provide publication information, including full citations and links to full text where available. Publication information may be manually catalogued through the VIVO UI, or come from your institutional repository or from index services (Crossref or PubMed). It is best to check with your local VIVO maintainers regarding their practices for updating your publications.

6.2.5.1 Harvesting publication metadata from Crossref and PubMed

There is a support for harvesting of publication information by providing DOI or PubMed ID for global index services Crossref and PubMed. To make this option available, a runtime.properties createAndLink.providers should be defined, and VIVO should be restarted after that.

```
createAndLink.providers = doi, pmid
```

Depending on the selected theme, the boxes for running this harvesting should be displayed in the research profile page. For instance, if the theme *wilma* is used, the green buttons are available at the right part of the screen. The user should provide DOI or PubMed ID and follow the instructions through wizards. Usually, a confirming of creation of new publications (if it doesn't exist in the system) and establishing links to researchers profiles are needed. Note that if the DOI or PMID is already present, then the claim interface will return the existing publication from VIVO, not the external metadata. The publication should be claimed for all co-authors to be properly linked in the graph.



Enter DOI:

You may enter one or more DOIs to match, and can be entered either as an ID or URL:

e.g.
ID: 10.1038/nature01234
URL: https://doi.org/10.1038/nature01234

Currently, DOIs issued by Crossref, DataCite and mEDRA are supported.
Each DOI should be separated by a comma or new line.

6.2.6 Additional Items for Your Profile

Other items on your profile can be added using similar techniques. You can add contact information, research areas, geographic focus, service to the profession, and focused overviews describing teaching, research and service. You may wish to check with your VIVO providers regarding local practices, which data are provided automatically, and which you may provide.

6.3 Using Search

Note

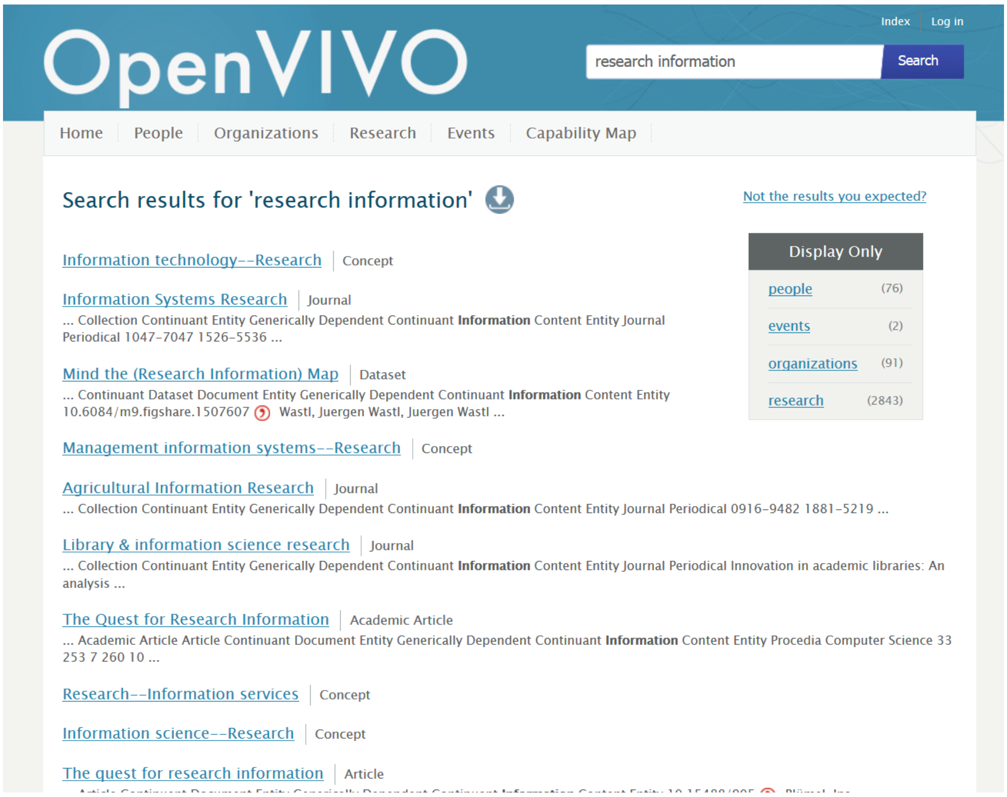
VIVO can be customized by each VIVO site to provide a "look and feel" as needed for each site. Some sites will move the search functionailty. The description below describes a "standard" VIVO, one in which customization is minimal. If the instructions below do not pertain to your local site, you may wish to inquire locally if documentation regarding the use of search and other VIVO functionality is available locally.

6.3.1 Searching

The search function allows you to discover the scholarship in your VIVO. In a standard VIVO there is a search box on the home page. The search box can also be found at the top of every page. Here you can enter search terms.

Search tips

- Keep it simple! Use short, single terms unless your searches are returning too many results.
- Use quotes to search for an entire phrase -- e.g., "*protein folding*".
- Except for boolean operators, searches are **not** case-sensitive, so "Geneva" and "geneva" are equivalent
- If you are unsure of the correct spelling, put ~ at the end of your search term -- e.g., *cabage~* finds *cabbage*, *steven~* finds *Stephen* and *Stefan* (as well as other similar names).



Search results for 'research information'

Information technology--Research | Concept

Information Systems Research | Journal
... Collection Continuant Entity Generically Dependent Continuant **Information** Content Entity Journal Periodical 1047-7047 1526-5536 ...

Mind the (Research Information) Map | Dataset
... Continuant Dataset Document Entity Generically Dependent Continuant **Information** Content Entity 10.6084/m9.figshare.1507607 Wastl, Juergen Wastl, Juergen Wastl ...

Management information systems--Research | Concept

Agricultural Information Research | Journal
... Collection Continuant Entity Generically Dependent Continuant **Information** Content Entity Journal Periodical 0916-9482 1881-5219 ...

Library & information science research | Journal
... Collection Continuant Entity Generically Dependent Continuant **Information** Content Entity Journal Periodical Innovation in academic libraries: An analysis ...

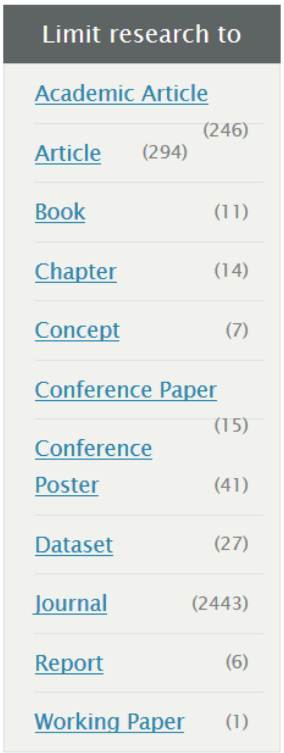
The Quest for Research Information | Academic Article
... Academic Article Article Continuant Document Entity Generically Dependent Continuant **Information** Content Entity Procedia Computer Science 33 253 7 260 10 ...

Research--Information services | Concept

Information science--Research | Concept

The quest for research information | Article

If you want to refine your search results, you can choose to display the results of a "facet" – a subset of information, all of the same type. The facets are shown on the right side. Selection of a facet limits your results to list items of the specified type. VIVO shows the number of items that will be returned for each facet. Below we see that selecting "Academic Article" as a facet will return 246 academic articles matching the search criteria.

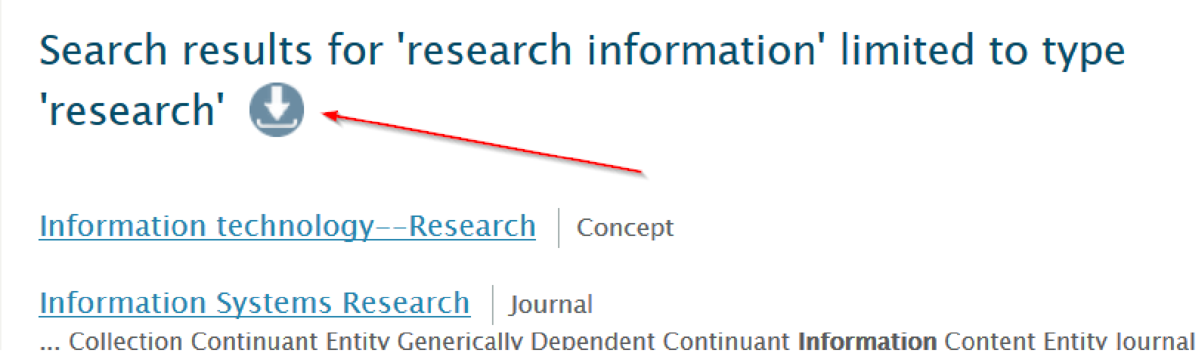


Limit research to

Academic Article	(246)
Article	(294)
Book	(11)
Chapter	(14)
Concept	(7)
Conference Paper	(15)
Conference Poster	(41)
Dataset	(27)
Journal	(2443)
Report	(6)
Working Paper	(1)

6.3.2 Search results

If you are satisfied with your search results you can click on one of the links or download the results. If you choose to do the latter, please click on the download symbol shown in the following figure.



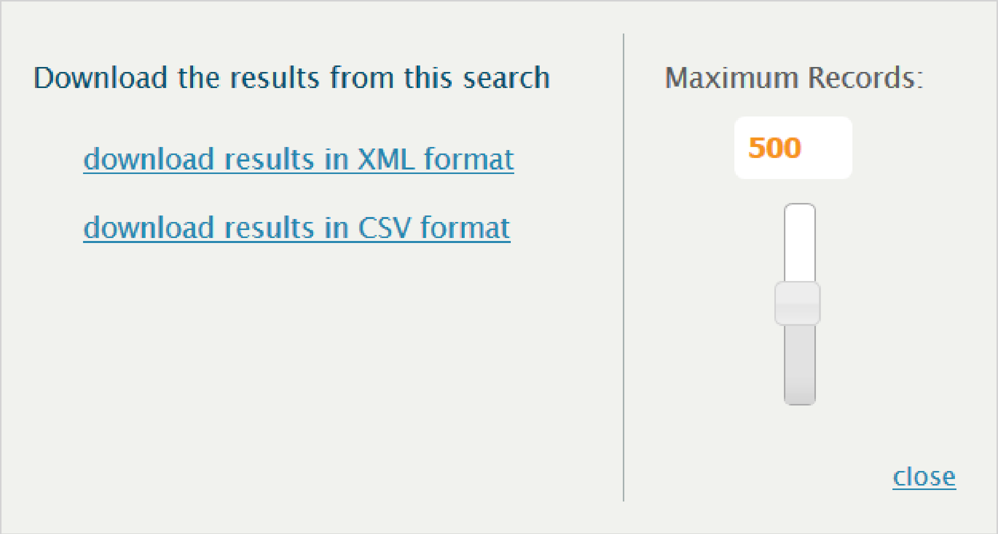
Search results for 'research information' limited to type 'research'

Information technology--Research | Concept

Information Systems Research | Journal

... Collection Continuant Entity Generically Dependent Continuant **Information** Content Entity Journal

VIVO allows you to download search results as XML or CSV file. You can change the number of records on the right side by using a slider. Download starts after clicking on "download results in * format".



Download the results from this search

download results in XML format

download results in CSV format

Maximum Records:

500

6.4 Using Visualizations

VIVO offers various possibilities to visualize the contained information. The default visualizations are presented below.

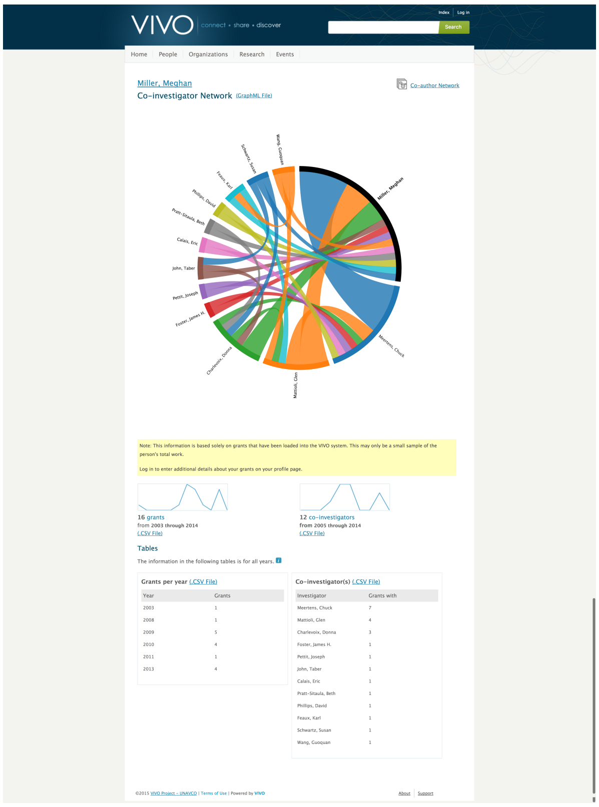
6.4.1 Visualizations in VIVO

- Capability Map (see page 89)
- Co-Authorship and Co-Investigator Network Visualizations (see page 87)
- Map of Science (see page 88)
- Temporal graph (see page 94)

6.4.2 Co-Authorship and Co-Investigator Network Visualizations

It is possible to display and investigate co-author and co-investigator networks in VIVO. Both tools share a common structure. The main component is a chord diagram, which is used to visualize the connection between co-authors or co-investigators. Note: This information is based solely on publications that have been loaded into the VIVO system. This may only be a small sample of the person's total work.

When you hover on the name of a person in the diagram, the visualization will focus on showing the connections of the selected person. Furthermore the number of joint publications (or grants) will be shown. Left-clicking on the name will load this person's VIVO profile.


VIVO
connect • share • discover
Index
Log in
Search
Home
People
Organizations
Research
Events
Co-author Network
Wang, Guoquan
Schwartz, Susan
Feaux, Karl
Phillips, David
Pratt-Sitaula, Beth
Calais, Eric
John, Taber
Pettit, Joseph
Foster, James H.
Charlevoix, Donna
Mattioli, Glen
Meertens, Chuck
Miller, Meghan

Miller, Meghan

Co-investigator Network (GraphML File)

Note: This information is based solely on grants that have been loaded into the VIVO system. This may only be a small sample of the person's total work.

Log in to enter additional details about your grants on your profile page.

16 grants
from 2003 through 2014
(.CSV File)

12 co-investigators
from 2005 through 2014
(.CSV File)

Tables

The information in the following tables is for all years.

Grants per year (.CSV File)

Year	Grants
2003	1
2008	1
2009	5
2010	4
2011	1
2013	4

Co-investigator(s) (.CSV File)

Investigator	Grants with
Meertens, Chuck	7
Mattioli, Glen	4
Charlevoix, Donna	3
Foster, James H.	1
Pettit, Joseph	1
John, Taber	1
Calais, Eric	1
Pratt-Sitaula, Beth	1
Phillips, David	1
Feaux, Karl	1
Schwartz, Susan	1
Wang, Guoquan	1

©2015 VIVO Project – UNAVCO | Terms of Use | Powered by VIVO
About | Support

Below the chord diagram you will find more tools. Firstly, two sparklines, one showing the number of publications of the person per year, the other the number of co-authors per year. It is possible to export the underlying data as csv files. Co-authors and publications per year will be shown in tables at the bottom of the page as well.

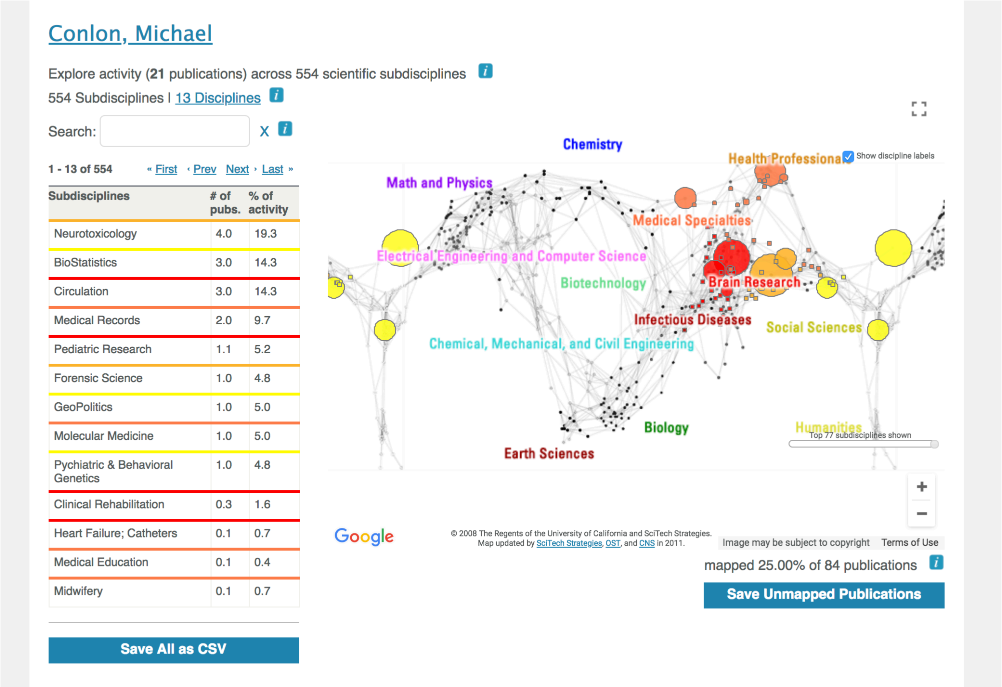
6.4.3 Map of Science

VIVO provides a representation of the research areas of a scholar as a "map of science". The underlying map is fixed, and represents the fields of science as determined by an analysis of more than 1 million journal articles. The size of the bubbles on the map indicates the relative number of papers in that area. The colors indicate the subject areas. The connections on on the map indicate the co-author tendencies of authors in each of the disciples. Disciplines are grouped on the map by co-author tendencies.

The Map of Science determines subject areas by matching publications of the author with journals in which the articles appeared, and using subject areas of the journals. In the lower right of the map we see that the Map of Science has mapped 25% of the author's publications. These publications were in journals whose subject areas are known to the Map of Science.

Overing over bubbles will display the name of the discipline.

Publication counts used by the map of science can be downloaded as a CSV from the panel on the left of the display. Unmapped publication venues can be downloaded from the panel on the right of the display.



Conlon, Michael

Explore activity (**21** publications) across 554 scientific subdisciplines

554 Subdisciplines | 13 Disciplines

Search:

1 - 13 of 554 « First ‹ Prev Next › Last »

Subdisciplines	# of pubs.	% of activity
Neurotoxicology	4.0	19.3
BioStatistics	3.0	14.3
Circulation	3.0	14.3
Medical Records	2.0	9.7
Pediatric Research	1.1	5.2
Forensic Science	1.0	4.8
GeoPolitics	1.0	5.0
Molecular Medicine	1.0	5.0
Pychiatric & Behavioral Genetics	1.0	4.8
Clinical Rehabilitation	0.3	1.6
Heart Failure; Catheters	0.1	0.7
Medical Education	0.1	0.4
Midwifery	0.1	0.7

Save All as CSV

mapped 25.00% of 84 publications

Save Unmapped Publications

Chemistry
Health Professionals
☑ Show discipline labels
Math and Physics
Medical Specialties
Electrical Engineering and Computer Science
Biotechnology
Brain Research
Infectious Diseases
Social Sciences
Chemical, Mechanical, and Civil Engineering
Biology
Humanities
Top 77 subdisciplines shown
Earth Sciences
Google
© 2008 The Regents of the University of California and SciTech Strategies. Map updated by SciTech Strategies, OST, and CNS in 2011.
Image may be subject to copyright Terms of Use

6.4.4 Using the Capability Map

6.4.4.1 Overview

The VIVO Capability Map provides a visual means for finding people working in particular areas and for navigating through concepts and people. Using the capability map, one might find people to invite to a workshop, colleagues for a grant application, members of a dissertation committee, or experts for a technical advisory group. The map is very easy to use.

The capability map was originally developed by a team at the University of Melbourne for their VIVO, Find an expert[64]. The concept was developed for OpenVIVO[65] and introduced to VIVO in version 1.9.0.

The capability map uses D3.js[66], a visualization tool used by the NY Times and others to provide modern graphics that are interactive, and responsive – that is, work on any device in any modern browser from a phone to a desktop computer.

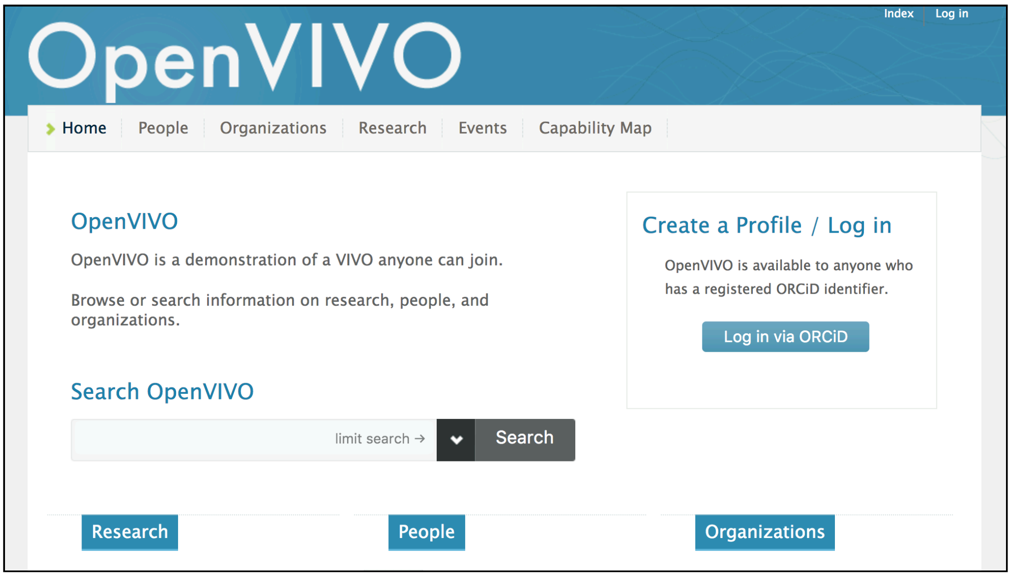
In the examples that follow, we will use OpenVIVO. If your local VIVO is at version 1.9.0 or higher, it will have the capability map as described here.

6.4.4.2 A Tour of the Capability Map

Think of a research area that interests you. Perhaps you study ontologies and want to find other people who are interested in ontology. Click on the Capability Map link in the menu bar on the VIVO home page (see below). You do not need to be logged on. If you are using your local VIVO and you do not see the capability map link on the home page, please contact your local VIVO support for more information.

---

[64] https://findanexpert.unimelb.edu.au/
[65] http://openvivo.org
[66] http://d3js.org


OpenVIVO
Index
Log in
Home
People
Organizations
Research
Events
Capability Map
Search
limit search →
Log in via ORCiD
Research
People
Organizations

OpenVIVO

OpenVIVO is a demonstration of a VIVO anyone can join.

Browse or search information on research, people, and organizations.

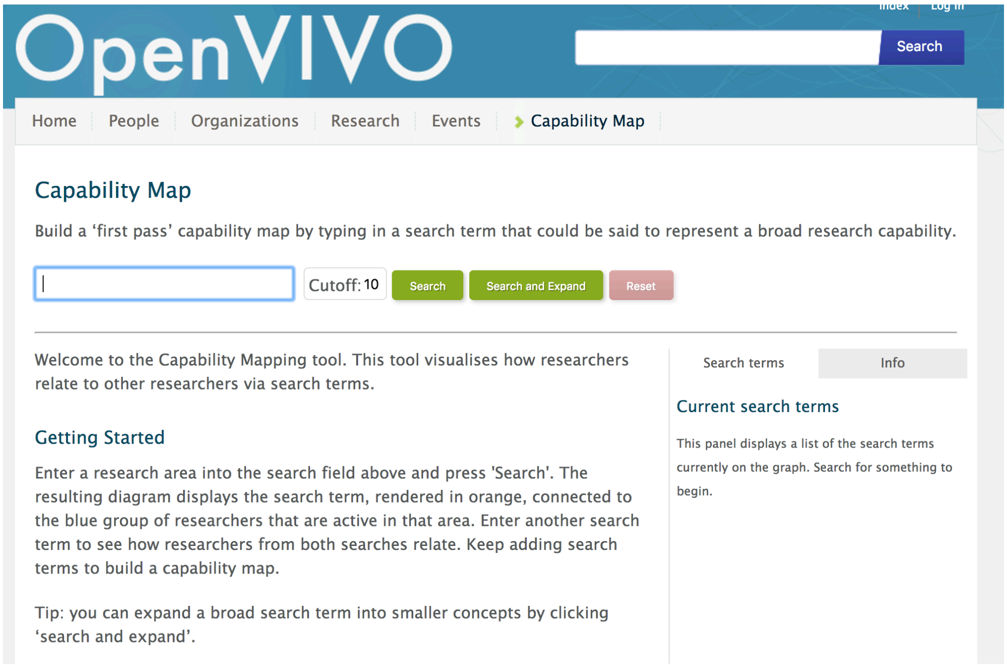
Search OpenVIVO

Create a Profile / Log in

OpenVIVO is available to anyone who has a registered ORCiD identifier.

1 Click on Capability Map on the VIVO home page

You will see the Capability Map page. Everything you do with the Capability Map will be done from this page. There are directions on the page. You'll want to read them all. Scroll down the page to see the rest of the directions.


OpenVIVO
Index
Log in
Search
Home
People
Organizations
Research
Events
Capability Map
Cutoff: 10
Search
Search and Expand
Reset
Search terms
Info

Capability Map

Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

Welcome to the Capability Mapping tool. This tool visualises how researchers relate to other researchers via search terms.

Getting Started

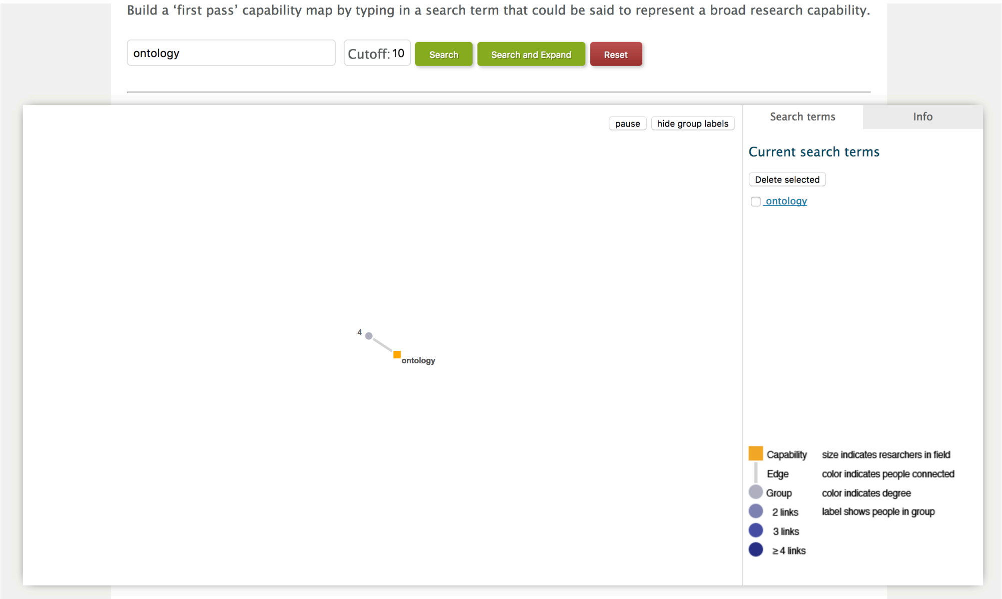
Enter a research area into the search field above and press 'Search'. The resulting diagram displays the search term, rendered in orange, connected to the blue group of researchers that are active in that area. Enter another search term to see how researchers from both searches relate. Keep adding search terms to build a capability map.

Tip: you can expand a broad search term into smaller concepts by clicking 'search and expand'.

Current search terms

This panel displays a list of the search terms currently on the graph. Search for something to begin.

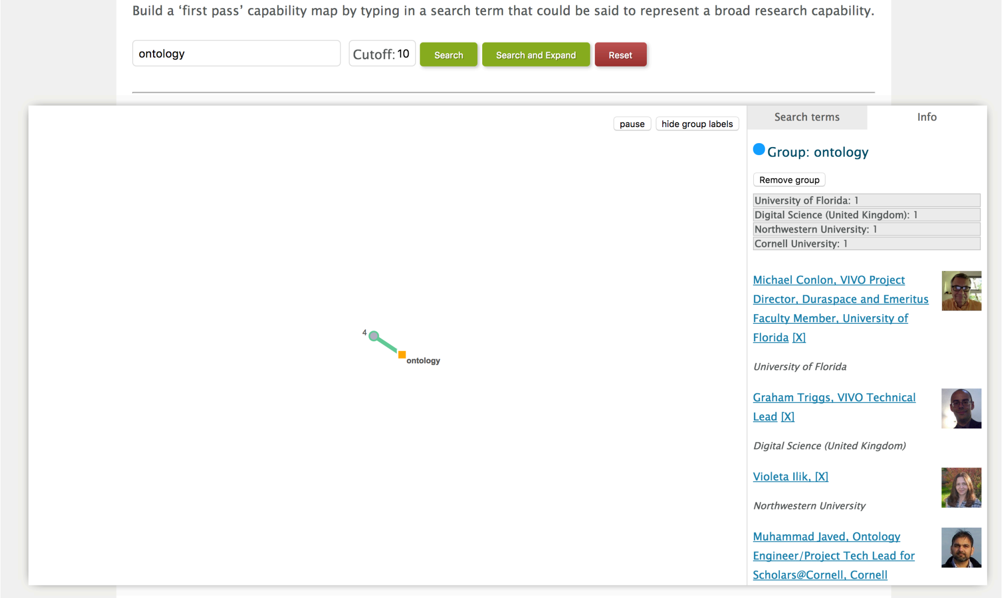
Type ontology into the text box and press Search. You should see something similar to below. OpenVIVO is a VIVO anyone can join. As a result, the data in OpenVIVO at the time you are following these examples may be different from the data at the time the examples were captured for this document.


Build a ‘first pass’ capability map by typing in a search term that could be said to represent a broad research capability.

ontology
Cutoff: 10
Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Current search terms
Delete selected
☐ ontology
4
ontology
Capability
size indicates resarchers in field
Edge
color indicates people connected
Group
color indicates degree
2 links
label shows people in group
3 links
≥ 4 links

Doesn't look like much, but we're just getting started. we have a very simple "graph" – a network diagram. The diagram has a legend in the lower right. There are three kinds of things on the graph: 1) terms or capabilities (orange squares); 2) groups of people (purple circles), and 3) edges (grey lines). Terms or capabilities are also called "research areas" or "research interests". They are concepts that are associated with individuals through works such as publications, grants, datasets and presentations, and through self-identification. People with VIVO profiles can select their research areas of interest using the profile editing features of VIVO. Each orange square represents a concept, and each is labeled. The purple dots represent groups of people. People are in the group if they have the research interests that connect to the group. In the figure above, there is just one concept – ontology – and just one group – the group of people with ontology as a research interest. Note the number of people in the group is shown (4). If there is just one person, the person's name will be shown. The edge indicates that the people in the group are "connected" to the term.

Click on the group of people. The right hand area indicates the people who are in the group. You can click on any of them to go to their VIVO page.


Build a ‘first pass’ capability map by typing in a search term that could be said to represent a broad research capability.

ontology
Cutoff: 10
Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
4
ontology

Group: ontology

Remove group

University of Florida: 1
Digital Science (United Kingdom): 1
Northwestern University: 1
Cornell University: 1

Michael Conlon, VIVO Project Director, Duraspace and Emeritus Faculty Member, University of Florida [X]

University of Florida

Graham Triggs, VIVO Technical Lead [X]

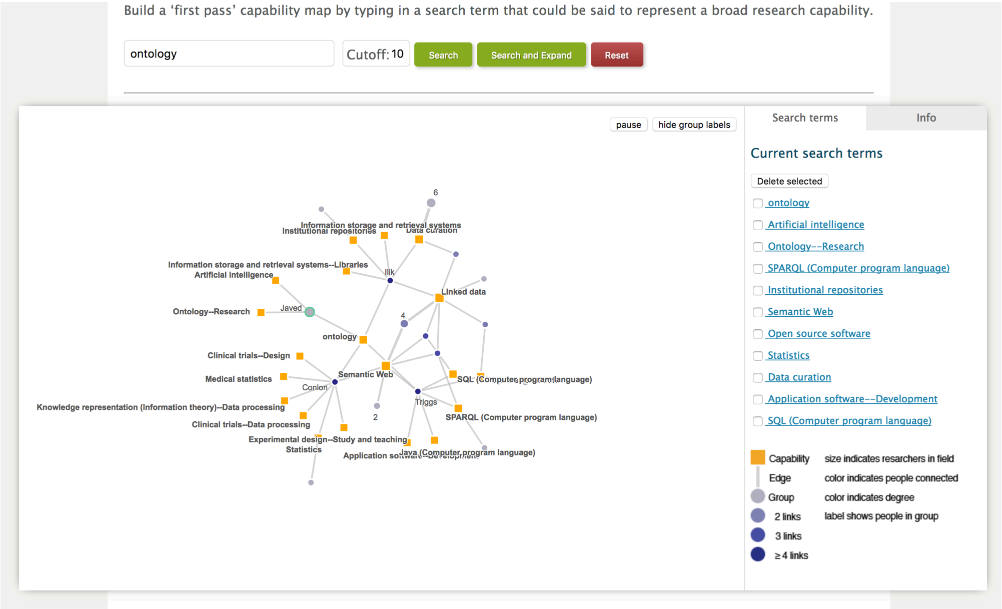
Digital Science (United Kingdom)

Violeta Ilik, [X]

Northwestern University

Muhammad Javed, Ontology Engineer/Project Tech Lead for Scholars@Cornell, Cornell

Let's add to the network. Click on Search and Expand. We now see the concepts of each of the four original people, with people associated with those concepts, connected in a graph showing common interests. Some people have many interests, some have just a few. Note that concepts are always connected to groups of people who have that concept has a research interest.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology

Cutoff: 10

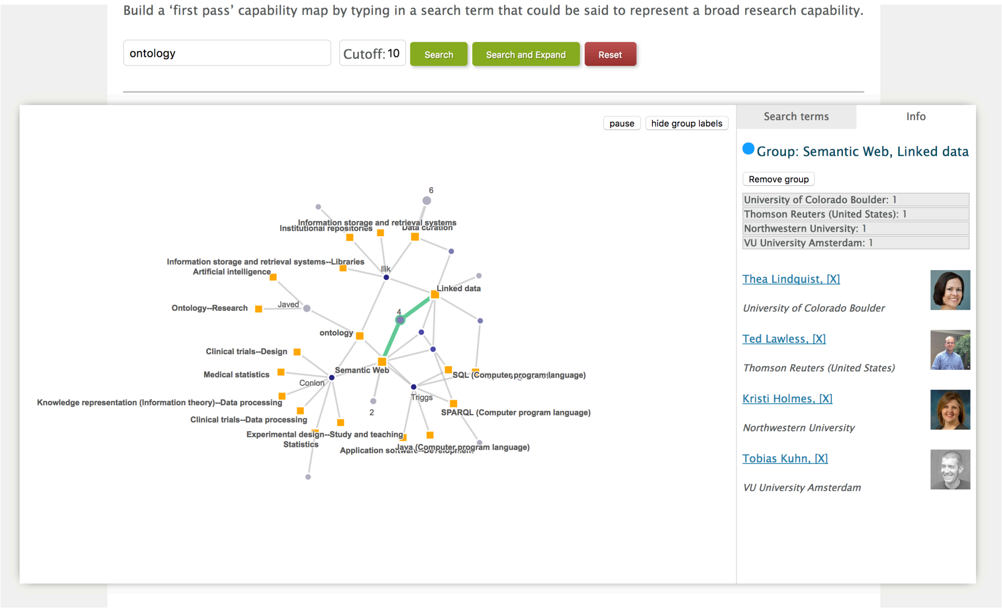
Current search terms

- ☐ ontology
- ☐ Artificial intelligence
- ☐ Ontology--Research
- ☐ SPARQL (Computer program language)
- ☐ Institutional repositories
- ☐ Semantic Web
- ☐ Open source software
- ☐ Statistics
- ☐ Data curation
- ☐ Application software--Development
- ☐ SQL (Computer program language)

🟧	Capability	size indicates resarchers in field
	Edge	color indicates people connected
●	Group	color indicates degree
●	2 links	label shows people in group
●	3 links	
●	≥4 links	

Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Delete selected
6
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Ilik
Linked data
Ontology--Research
Javed
4
ontology
Clinical trials--Design
Medical statistics
Semantic Web
Conlon
SQL (Computer program language)
Triggs
Knowledge representation (Information theory)--Data processing
2
SPARQL (Computer program language)
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Application software--Development
Java (Computer program language)

Click on the group of people near the center of the map, between linked data and semantic web. You should see a display similar to the one below. The group is highlighted along with the concepts common to the group (linked data and semantic web). Not that the right hand display shows the group as being defined as those people who are interested in both linked data and semantic web.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology

Cutoff: 10

Group: Semantic Web, Linked data

University of Colorado Boulder: 1
Thomson Reuters (United States): 1
Northwestern University: 1
VU University Amsterdam: 1

Thea Lindquist, [X]
University of Colorado Boulder

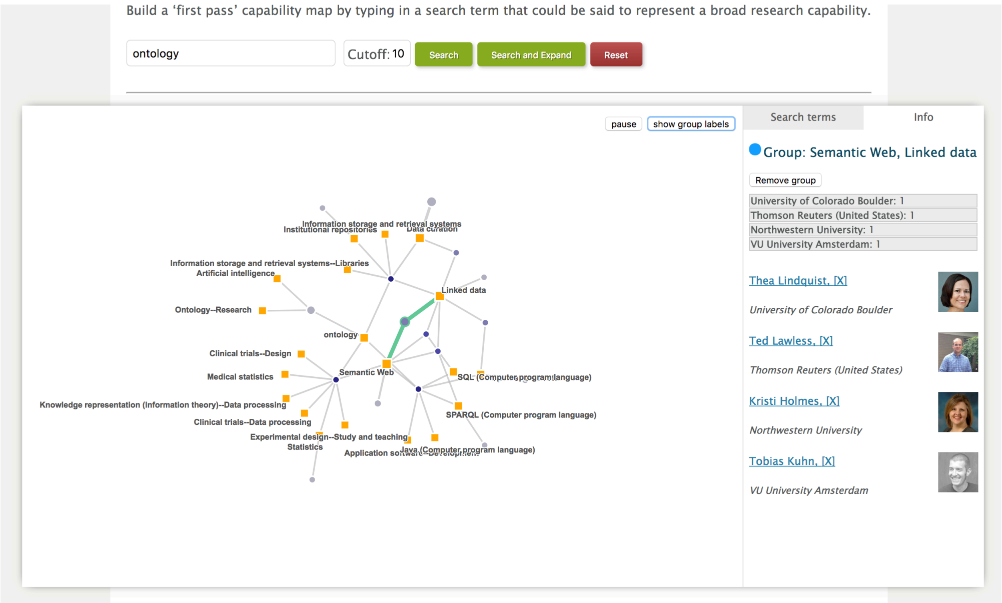
Ted Lawless, [X]
Thomson Reuters (United States)

Kristi Holmes, [X]
Northwestern University

Tobias Kuhn, [X]
VU University Amsterdam

Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Remove group
6
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Ilik
Linked data
Ontology--Research
Javed
4
ontology
Clinical trials--Design
Medical statistics
Semantic Web
Conlon
SQL (Computer program language)
Triggs
Knowledge representation (Information theory)--Data processing
2
SPARQL (Computer program language)
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Application software--Development
Java (Computer program language)

The map is a little cluttered with the names of the people who are in groups with one member. Click hide group labels. You see a map similar to the one below.



Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology

Cutoff: 10

Search
Search and Expand
Reset
pause
show group labels
Search terms
Info
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Linked data
Ontology--Research
ontology
Clinical trials--Design
Semantic Web
Medical statistics
SQL (Computer program language)
Knowledge representation (Information theory)--Data processing
Clinical trials--Data processing
SPARQL (Computer program language)
Experimental design--Study and teaching
Statistics
Java (Computer program language)
Application software--Development
Remove group

Group: Semantic Web, Linked data

University of Colorado Boulder: 1
Thomson Reuters (United States): 1
Northwestern University: 1
VU University Amsterdam: 1

Thea Lindquist, [X]

University of Colorado Boulder

Ted Lawless, [X]

Thomson Reuters (United States)

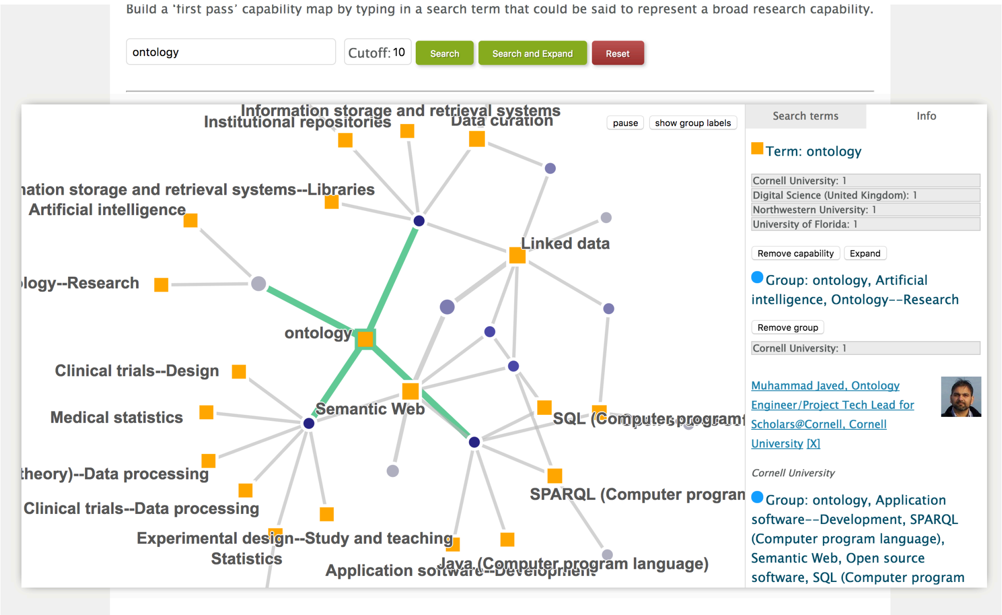
Kristi Holmes, [X]

Northwestern University

Tobias Kuhn, [X]

VU University Amsterdam

Double click on ontology. Two things happen. The map zooms in one ontology, and the ontology term is highlighted. One of the groups connected to ontology is highlighted.



Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology

Cutoff: 10

Search
Search and Expand
Reset
pause
show group labels
Search terms
Info
Information storage and retrieval systems
Institutional repositories
Data curation
ation storage and retrieval systems--Libraries
Artificial intelligence
Linked data
ology--Research
ontology
Clinical trials--Design
Semantic Web
Medical statistics
SQL (Computer program
theory)--Data processing
SPARQL (Computer progran
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Java (Computer program language)
Application software--Development
Remove capability
Expand
Remove group

Term: ontology

Cornell University: 1
Digital Science (United Kingdom): 1
Northwestern University: 1
University of Florida: 1

Group: ontology, Artificial intelligence, Ontology--Research

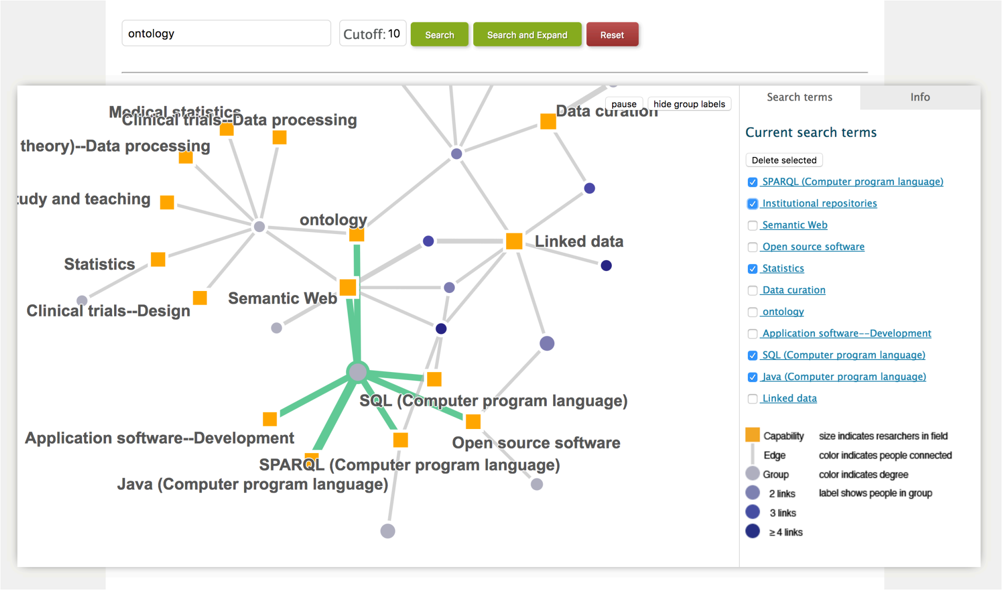
Cornell University: 1

Muhammad Javed, Ontology Engineer/Project Tech Lead for Scholars@Cornell, Cornell University [X]

Cornell University

Group: ontology, Application software--Development, SPARQL (Computer program language), Semantic Web, Open source software, SQL (Computer program

suppose we wish to reduce the number of terms that have been generated by expanding the graph – some of these are closely related to ontology, while others may be less so. Terms aon on the graph because someone who has ontology an an interest has one of these terms as an additional interest. Click on SQL (Computer Program Language), then in the right hand panel, click on "Search Terms" This lists all the terms currently in the graph.



ontology

Cutoff: 10

Search terms

Current search terms

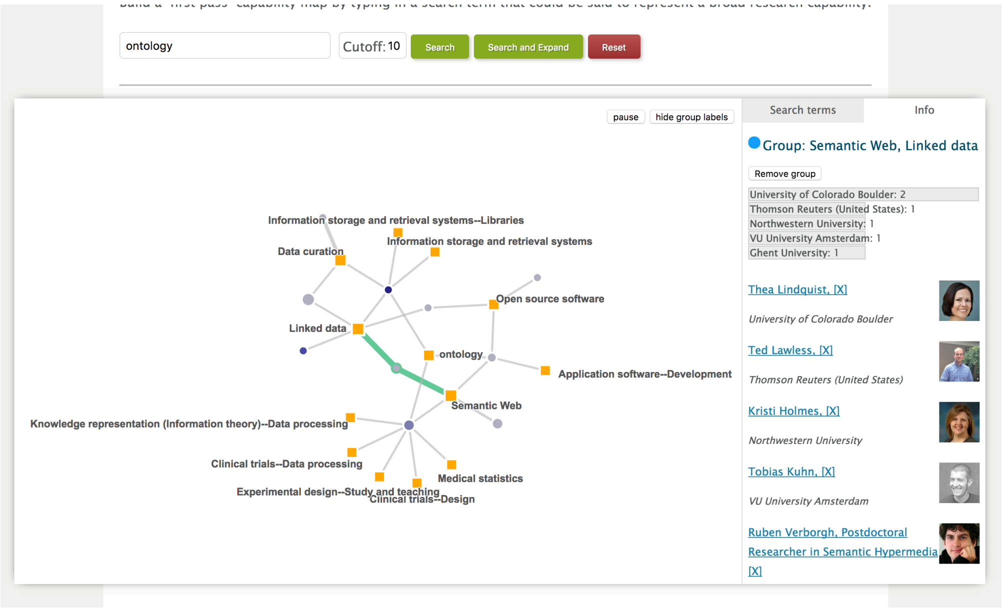
Delete selected

- ☑ SPARQL (Computer program language)
- ☑ Institutional repositories
- ☐ Semantic Web
- ☐ Open source software
- ☑ Statistics
- ☐ Data curation
- ☐ ontology
- ☐ Application software--Development
- ☑ SQL (Computer program language)
- ☑ Java (Computer program language)
- ☐ Linked data

🟧	Capability	size indicates researchers in field
	Edge	color indicates people connected
●	Group	color indicates degree
●	2 links	label shows people in group
●	3 links	
●	≥ 4 links	

Search
Search and Expand
Reset
pause
hide group labels
Info
Medical statistics
Clinical trials--Data processing
theory)--Data processing
tudy and teaching
Statistics
Clinical trials--Design
ontology
Semantic Web
Data curation
Linked data
SQL (Computer program language)
Application software--Development
Open source software
SPARQL (Computer program language)
Java (Computer program language)

Select five of the terms as shown and press "Delete Selected" The pruned graph is shown below. We seem more clearly the group of people interested in linked data and the semantic web.



build a "first pass" capability map by typing in a search term that could be said to represent a broad research capability.

ontology

Cutoff: 10

Group: Semantic Web, Linked data

Remove group

University of Colorado Boulder: 2	
Thomson Reuters (United States): 1	
Northwestern University: 1	
VU University Amsterdam: 1	
Ghent University: 1	

Thea Lindquist, [X]
University of Colorado Boulder

Ted Lawless, [X]
Thomson Reuters (United States)

Kristi Holmes, [X]
Northwestern University

Tobias Kuhn, [X]
VU University Amsterdam

Ruben Verborgh, Postdoctoral Researcher in Semantic Hypermedia [X]

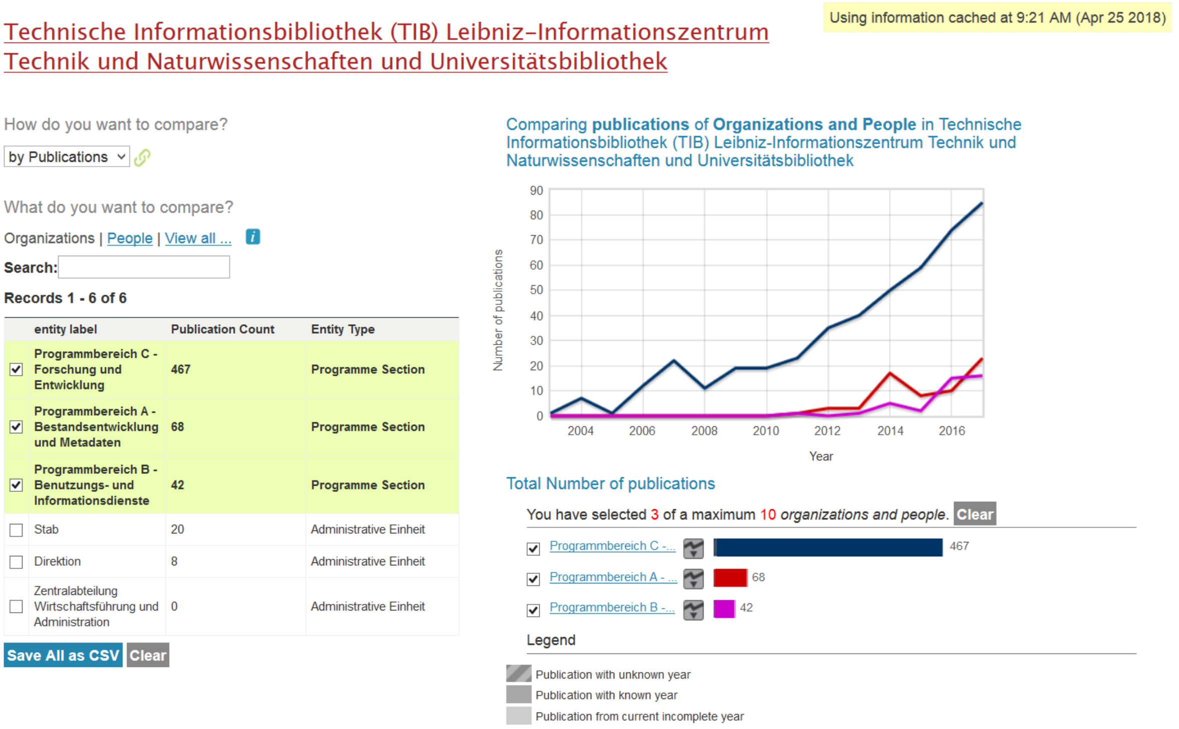
Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Information storage and retrieval systems--Libraries
Information storage and retrieval systems
Data curation
Open source software
Linked data
ontology
Application software--Development
Semantic Web
Knowledge representation (Information theory)--Data processing
Clinical trials--Data processing
Medical statistics
Experimental design--Study and teaching
Clinical trials--Design

We can search and expand at any time to add groups and terms related to those on the graph. And if we want to start over, we can press Reset.

Using the Capability Map we can find groups of people interest in common research areas, and we can discover which research areas are often held in common across researchers.

6.4.5 Using the Temporal Graph

The temporal graph is one of the standard visualizations provided by VIVO. It allows one to compare up to ten persons or organizations on a time axis. You can select if you want to compare by publications, or by grants.



Technische Informationsbibliothek (TIB) Leibniz-Informationszentrum Technik und Naturwissenschaften und Universitätsbibliothek

Using information cached at 9:21 AM (Apr 25 2018)

How do you want to compare?

by Publications

What do you want to compare?

Organizations | People | View all ...

Search:

Records 1 - 6 of 6

	entity label	Publication Count	Entity Type
☑	**Programmbereich C - Forschung und Entwicklung**	**467**	**Programme Section**
☑	**Programmbereich A - Bestandsentwicklung und Metadaten**	**68**	**Programme Section**
☑	**Programmbereich B - Benutzungs- und Informationsdienste**	**42**	**Programme Section**
☐	Stab	20	Administrative Einheit
☐	Direktion	8	Administrative Einheit
☐	Zentralabteilung Wirtschaftsführung und Administration	0	Administrative Einheit

Save All as CSV | Clear

Comparing **publications** of **Organizations and People** in Technische Informationsbibliothek (TIB) Leibniz-Informationszentrum Technik und Naturwissenschaften und Universitätsbibliothek

Total Number of publications

You have selected 3 of a maximum 10 *organizations and people.* Clear

☑	Programmbereich C -...	467
☑	Programmbereich A -...	68
☑	Programmbereich B -...	42

Legend

Publication with unknown year
Publication with known year
Publication from current incomplete year

The name of the organization whose output is displayed is shown above the visualization. On the right side in a yellow box you can read when the information shown here was last updated. On the left side you can now set whether publications or grants should be compared with each other and whether these comparisons should refer to persons, organisations or both together. Only the ten entries at a time are displayed. Further entities can be added via search slot.

If an entity is added to the comparison, the output is visualized on the right side in the temporal graph. Under the display you can remove the selected entities individually or altogether ("clear all").

It is also possible to export the number and type of publications per person and entity ("Save All as CSV").

6.5 VIVO for Data Analysts

6.5.1 Background

VIVO represents data as triples. All data is represented and stored in the form subject, predicate, object. All entities are identified by URI. The W3C has developed standards for RDF (Resource Description Framework) for such representations and for various serializations of RDF, including Turtle. If you are unfamiliar with this method for data representation, see the references. A typical VIVO for a large research institution could have well over 10 million triples. The triples are defined using an ontology. The ontology is described here: Ontology Reference (see page 446) Understanding which triples are needed for an analysis can be challenging. The VIVO community is here to help. Questions regarding data and data extraction using the techniques below can be posted to one of the VIVO Google Groups[67]. You may also wish to contact the VIVO providers at your institution who may be able to help with some of the technologies involved.

---

[67] https://wiki.duraspace.org/display/VIVO/Email+Lists

6.5.2 Getting Rectangles of Data

To get rectangles of data, use SPARQL queries. SPARQL is a simple query language designed for use with triple stores. Use of SPARQL is described here: SPARQL Queries (see page 118)

6.5.3 Getting Graphs of Data

The entire triple store can be unloaded for use in a local triple store, and for local query. This is recommended for sites wishing to make repeated analyst queries of the data. Community-editions of a triple stores are available with no cost. Stardog is a popular, stable, and free triple store that can be used for this purpose. See http://stardog.com

To unload the triple store to a set of triples, use jena3tools, available here: https://github.com/vivo-project/jenatools

6.5.4 Repeatable Processes

To get data from VIVO on a regular basis, you may wish to work with your VIVO providers to create an API that you can use that will provide required data. The Data Distribution API (see page 339) is designed for this purpose and can be configured to return specified data, including parameterized data via configurable addresses in configurable data formats.

6.5.5 Distributed Queries

Some applications involve getting data from multiple VIVOs. VIVOs running version 1.10 and above provide a Triple Pattern Fragments (see page 380) endpoint which can be used to rapidly get all triples from a VIVO matching a triple pattern.

6.5.6 References

1. RDF 1.1 Primer https://www.w3.org/TR/rdf11-primer/
2. RDF 1.1 Turtle https://www.w3.org/TR/turtle/
3. Börner, Conlon, Corson-Rikert, and Ding (eds) VIVO: A Semantic Approach to Scholarly Networking and Discovery, Morgan-Claypool Publishers, 2012. 160 pages.
4. Allemang, and Hendler. Semantic Web for the Working Ontologist, second edition. Morgan-Kaufmann Publishers, 2011. 354 pages.
5. DuCharme. Learning SPARQL: Querying and Updating with SPARQL 1.1. O'Reilly, 2011. 235 pages.

7 Managing Data in Your VIVO

- Importing Data to VIVO (see page 97)
- Exporting Data from VIVO (see page 109)
- Managing Person Identifiers (see page 109)
- Managing Organization Hierarchy (see page 111)
- Managing Data Packages (see page 116)
- SPARQL Queries (see page 118)
- How to remove data from a specific graph (see page 123)
- Removing Entities from VIVO (see page 123)
- Uploading files associated with individuals (see page 126)

7.1 Importing Data to VIVO

- Using the Convert CSV to RDF ingest tool (see page 97)
- Data types for string and language (see page 107)

7.1.1 Using the Convert CSV to RDF ingest tool

This guide describes just one (rather primitive) way to ingest data into VIVO. See https://wiki.duraspace.org/x/MYUQAg for a detailed discussion of data ingest.

Introduction

This guide will walk through the use of the 'Convert CSV to RDF' tool, a semi-automated method of converting comma separated or tab separated text files into RDF that can be displayed in VIVO. These files should include one row of data per record (e.g., a person or publication) and represent the fields or properties associated with each record in separate columns within the row, much as the values appear in a spreadsheet. The most common pattern of loading CSV files involves one CSV file per type of data to be loaded. Note, the current ingest tools involve working through a number of steps from original source data files to the appearance of new data in VIVO. The process requires some understanding of semantic web data modeling and some training.

Access the tool by navigating to the Site Administration section, clicking Ingest tools, then Convert CSV to RDF.

7.1.1.1 **Mapping Ontologies to other Ontologies**

When VIVO's ingest tools read in a CSV file, the data read from the file will be stored in the VIVO database as an extra "model," or secondary database managed by the Jena semantic web libraries underlying VIVO.

The next step is to link imported data sets using information stored with the source data. If an object not previously in VIVO has been found, a new record (individual) must be created in VIVO using the CONSTRUCT form of query in the SPARQL query language for RDF.

The query looks at the list of imported data and inserts statements creating a new individual wherever no match to an existing individual is found. The query of imported data is expressed using the ontology of the import; the CONSTRUCT statements use the class and property names of VIVO ontology. This accomplishes the mapping between the source ontology and the VIVO ontology.

When populating VIVO for the first time, all the data from a dataset can be added from the imported Jena model to VIVO by a CONSTRUCT query that again translates from the RDF in the imported model to the RDF in VIVO. When a dataset has already been mapped to data and there's a likelihood that the new data being imported may already exist in VIVO, the process becomes more complicated due to the need to match each prospective import against existing data.

7.1.1.2 **Example workflow**

The first time through an ingest process is a slow process of evaluating the source data, deciding what cleanup will be needed, and working through each step. The following example details each step in the process and how the VIVO software supports that step. As the process expands to include requirements to match against existing data, additional steps must be introduced to test for matches and perform different actions, usually as successive steps identified through different queries.

For this example, a sample set of data for people, their positions, and the organizations at which their positions reside has been provided at http://sourceforge.net/projects/vivo/files/Data%20Ingest/. The guide gives instructions on how to ingest the data, map the data to the VIVO/ISF ontology, and load the data into an RDF format. Upon completion linked data in VIVO will include people to positions and the positions to organizations.

7.1.1.2.1 Process:

1. Prepare CSV file
2. Create workspace models for ingesting and constructing data
3. Pull CSV file into RDF
4. Confirm data property URIs and RDF structure
5. Convert temporary RDF into VIVO/ISF ontology using SPARQL CONSTRUCT
6. Load data to the current web model

7.1.1.2.2 *Step 1*: Prepare CSV file

CSV template files can be downloaded here (or from the Source Forge links included):

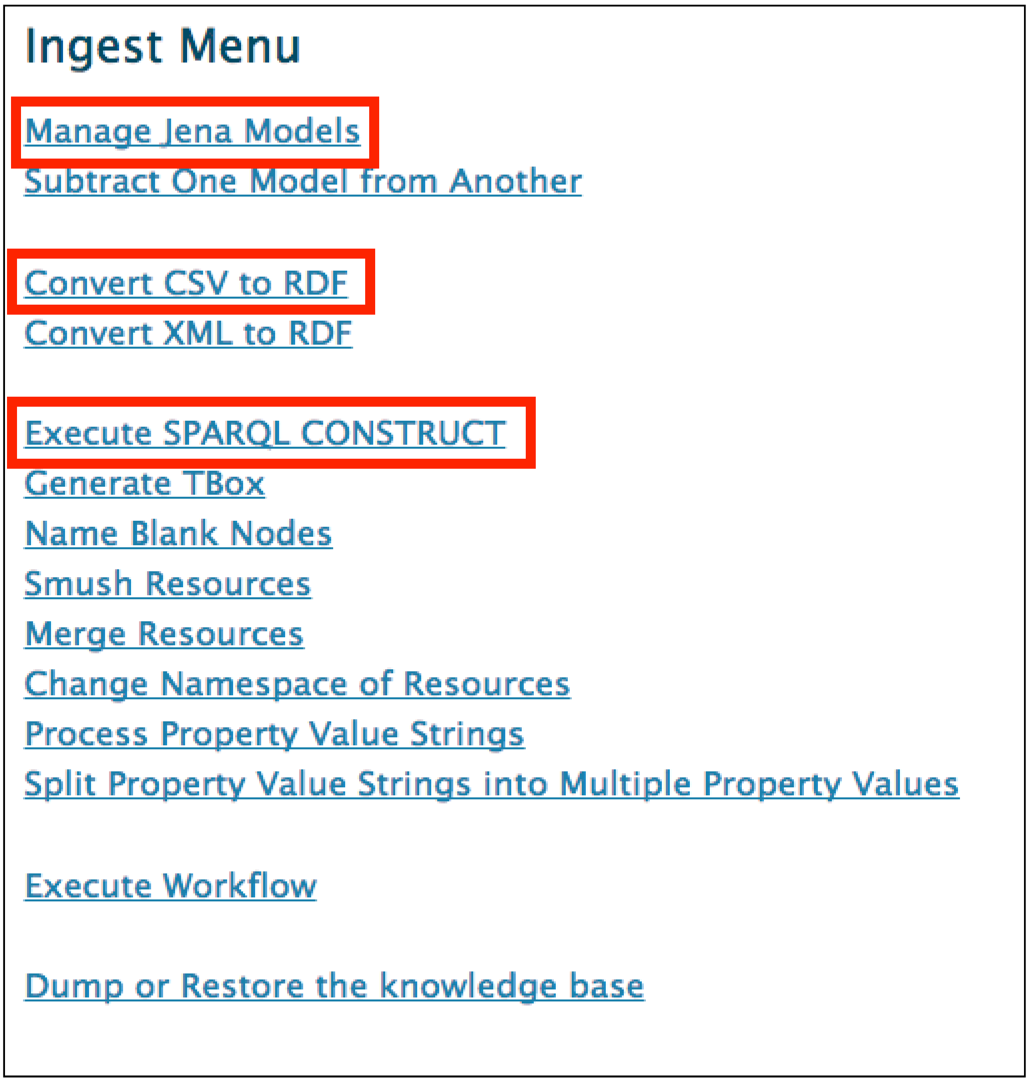
- organization.csv[68] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/organization.csv/download)
- position.csv[69] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/position.csv/download)
- people.csv[70] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/people.csv/download)

For the purposes of this walkthrough, you can leave the csv files as is if you wish.

Note: You can optionally create a local ontology and class specifically for the 'person_ID' and 'org_ID' fields included in the example CSV files. See the 'Add New Data Properties' section of the Ontology Editors Guide here: https://wiki.duraspace.org/x/2AQGAg.

7.1.1.2.3 *Step 2: Create Workspace Models*

Highlighted in red are the three Ingest Menu options we will be using for this demonstration.



Ingest Menu

Manage Jena Models
Subtract One Model from Another

Convert CSV to RDF
Convert XML to RDF

Execute SPARQL CONSTRUCT
Generate TBox
Name Blank Nodes
Smush Resources
Merge Resources
Change Namespace of Resources
Process Property Value Strings
Split Property Value Strings into Multiple Property Values

Execute Workflow

Dump or Restore the knowledge base

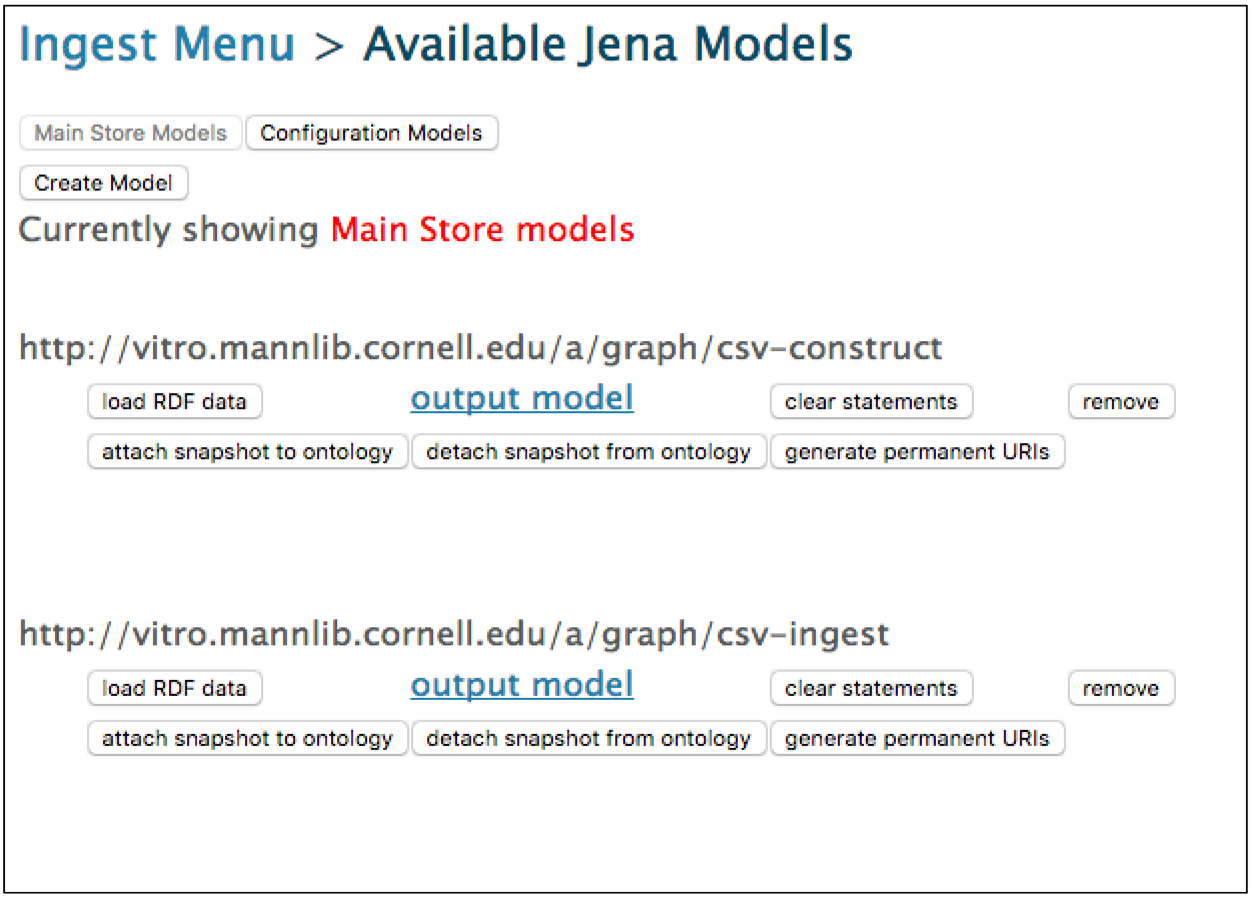
---

[68] https://wiki.lyrasis.org/download/attachments/260898919/organization.csv?api=v2&modificationDate=1664883350669&version=1
[69] https://wiki.lyrasis.org/download/attachments/260898919/position.csv?api=v2&modificationDate=1664883350657&version=1
[70] https://wiki.lyrasis.org/download/attachments/260898919/people.csv?api=v2&modificationDate=1664883350663&version=1

We will create two temporary data models titled 'csv-ingest' and 'csv-construct' to keep our work separate from the main VIVO models.

1. Select "Ingest Tools" from the Advanced Tools Menu
2. Select "Manage Jena Models"
3. Click the "Create Model" button then type in a name for your model, 'csv-ingest'.
4. Repeat step 3 and name the model 'csv-construct'

You should now see your new models on the main 'Manage Jena Models' page.



Ingest Menu > Available Jena Models

Currently showing Main Store models

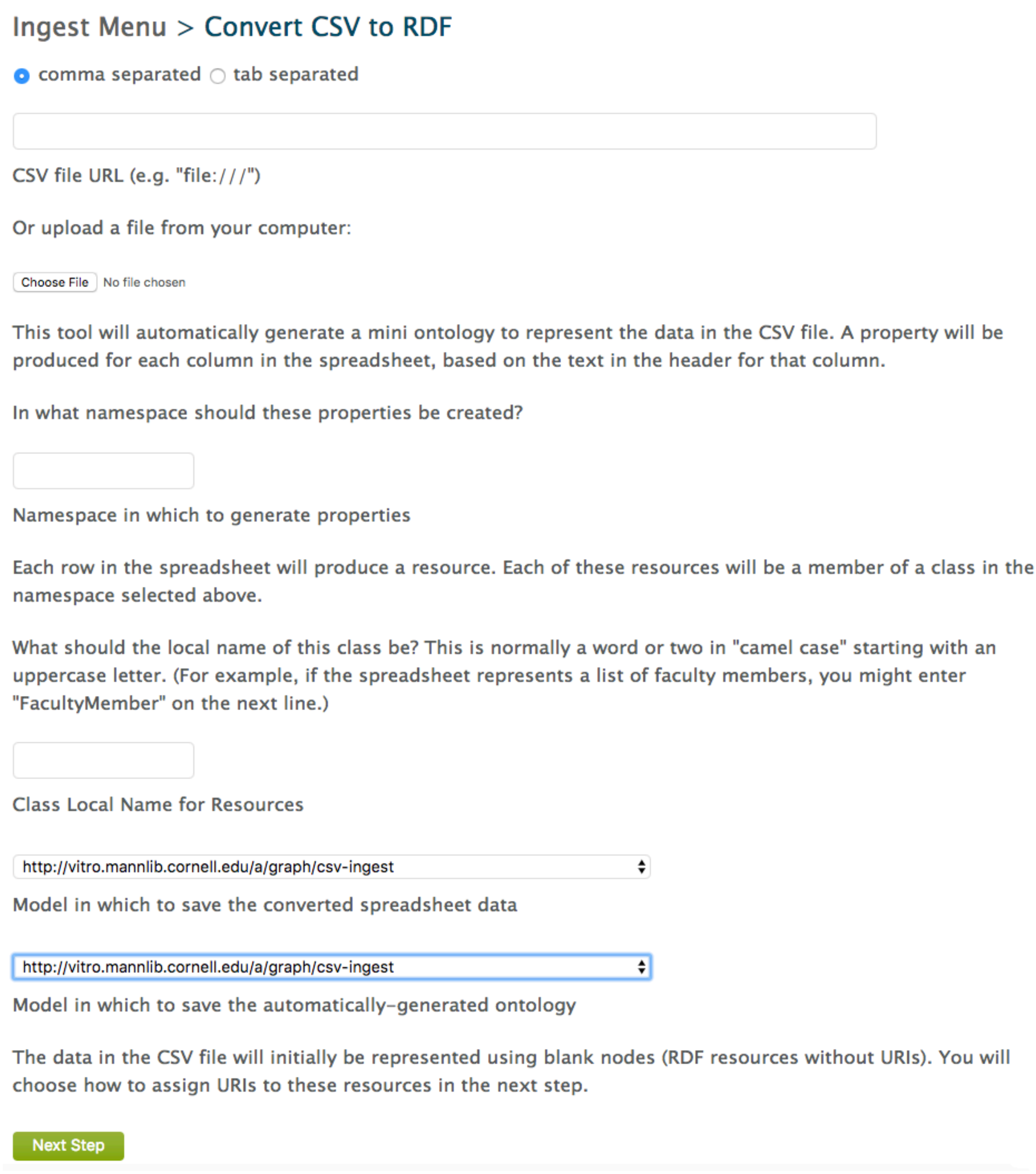
http://vitro.mannlib.cornell.edu/a/graph/csv-construct

http://vitro.mannlib.cornell.edu/a/graph/csv-ingest

Main Store Models
Configuration Models
Create Model
load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs

7.1.1.2.4 *Step 3*: Pull CSV File into RDF

Now click 'Convert CSV to RDF' on the Ingest Menu. Begin by supplying a URL for your CSV file, or by uploading the file directly from your computer. Start with people.csv. Complete the fields in the form (explanation follows graphic below).



Ingest Menu > Convert CSV to RDF

● comma separated ○ tab separated

CSV file URL (e.g. "file:///")

Or upload a file from your computer:

This tool will automatically generate a mini ontology to represent the data in the CSV file. A property will be produced for each column in the spreadsheet, based on the text in the header for that column.

In what namespace should these properties be created?

Namespace in which to generate properties

Each row in the spreadsheet will produce a resource. Each of these resources will be a member of a class in the namespace selected above.

What should the local name of this class be? This is normally a word or two in "camel case" starting with an uppercase letter. (For example, if the spreadsheet represents a list of faculty members, you might enter "FacultyMember" on the next line.)

Class Local Name for Resources

http://vitro.mannlib.cornell.edu/a/graph/csv-ingest

Model in which to save the converted spreadsheet data

http://vitro.mannlib.cornell.edu/a/graph/csv-ingest

Model in which to save the automatically-generated ontology

The data in the CSV file will initially be represented using blank nodes (RDF resources without URIs). You will choose how to assign URIs to these resources in the next step.

7.1.1.2.4.1 Namespace for Classes and Properties ('in what namespace should these properties be created?'

This namespace can be temporary since we will later map the tool's output to the VIVO/ISF ontology. For example, you can use

```
http://localhost/vivo/
```

Class Name

The class name is also a temporary value for this example. This value does not follow the created entity since you will shift the properties from the format they come in into the ontologies format. For this example, the suggested class names are “ws_ppl”, “ws_org”, and “ws_post”.

7.1.1.2.4.2 Destination Models

The data and ontology model option dropdown menus should list the model to ingest into. This is where we select one of those 'workspace' models we created earlier, "csv-ingest" for example.

After clicking convert we can check our RDF conversion through two methods, the first method being the quickest.

1. From the Ingest Menu, select the 'Manage Jena Models' and then select the ingest model's 'output model.' This is something you can open with WordPad and see the created triples for your CSV file.
2. Also from the Manage Jena Model we can attach the ingest model to the current webapp. Then we can go back to the Site Admin, select Class Hierarchy link, select 'All Classes' and navigate to the Class Name you created (individual, for example).

Click 'Next Step.' Here you will create the URI for your new individuals.

Select URI prefix

It is most convenient if this matches the URL for your VIVO instance plus '/individual/', e.g. http://vivo.university.edu/individual/.

URI suffix

This can be a random number or a created based on a pattern plus the value from your CSV file.

Now, click 'Convert CSV.' The CSV data should now be converted into temporary RDF and inserted into the 'csv-ingest' model.

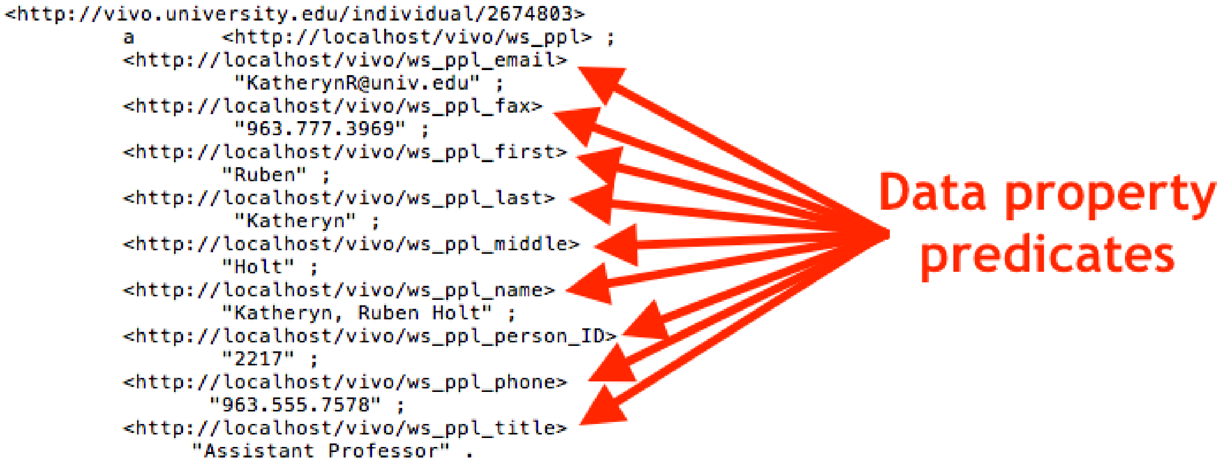
You can now repeat step 3 for both organizations.csv and positions.csv before continuing to assure positions will be attached to both people and organizations, or for simplicity, you can ignore organizations and positions for now.

7.1.1.2.5 *Step 4*: Confirm data property URIs and RDF structure

The CSV to RDF tool converts the CSV into temporary RDF that we can query using SPARQL. This temporary RDF cannot be displayed in VIVO. We will transform the RDF into the VIVO/ISF ontology format in the next step. First, take a look at the temporary RDF we have created.

7.1.1.2.5.1 Ingested Data URIs

Confirm the data property URI's that were created for each of the columns in your csv file by navigating back to the 'Manage Jena Models' page and clicking 'output model' below csv-ingest. A description of the format is described in the figure below. The predicates are the properties we need for our SPARQL Query. If you used the 'http://localhost/vivo/' (see page 97) format used above, it should look similar to this:



```
<http://vivo.university.edu/individual/2674803>
         a       <http://localhost/vivo/ws_ppl> ;
         <http://localhost/vivo/ws_ppl_email>
                   "KatherynR@univ.edu" ;
         <http://localhost/vivo/ws_ppl_fax>
                  "963.777.3969" ;
         <http://localhost/vivo/ws_ppl_first>
                 "Ruben" ;
         <http://localhost/vivo/ws_ppl_last>
                 "Katheryn" ;
         <http://localhost/vivo/ws_ppl_middle>
                 "Holt" ;
         <http://localhost/vivo/ws_ppl_name>
                 "Katheryn, Ruben Holt" ;
         <http://localhost/vivo/ws_ppl_person_ID>
                 "2217" ;
         <http://localhost/vivo/ws_ppl_phone>
                "963.555.7578" ;
         <http://localhost/vivo/ws_ppl_title>
              "Assistant Professor" .
```

We will use these data property predicates (e.g. <http://localhost/vivo/ws_ppl_email>) in the 'WHERE' part of the next step.

7.1.1.2.6 *Step 5*: Convert temporary RDF into VIVO/ISF ontology using SPARQL CONSTRUCT

Now, we must query and CONSTRUCT new RDF that is mapped to the VIVO/ISF ontology. Diagrams to help visualize the VIVO/ISF ontology model are available on the wiki at https://wiki.duraspace.org/x/ycCdB. Some helpful links to information on SPARQL queries can be found at https://wiki.duraspace.org/x/lwUGAg. Basically, we will query the database for the triples in Step 4 to pull out the resource URIs and store them in variables (e.g. http://vivo.university.edu/individual/2674803 → ?person) in the WHERE part of the query. We then CONSTRUCT new triples that are in the proper VIVO/ISF format using those variables for the resource URIs (e.g. ?person) and data values (e.g. ?fullname).

Return to the ingest menu and click 'Execute SPARQL CONSTRUCT'
Example SPARQL query for people.csv:

```
CONSTRUCT {

?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#FacultyMember> .
?person <http://www.w3.org/2000/01/rdf-schema#label> ?fullname .

?person <http://purl.obolibrary.org/obo/ARG_2000028> ?vcard .
?vcard <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Individual> .
?vcard <http://www.w3.org/2006/vcard/ns#hasName> ?vcard_name .
?vcard_name <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Name> .
?vcard_name <http://www.w3.org/2006/vcard/ns#givenName> ?first .
?vcard_name <http://vivoweb.org/ontology/core#middleName> ?middle .
?vcard_name <http://www.w3.org/2006/vcard/ns#familyName> ?last .

?vcard <http://www.w3.org/2006/vcard/ns#hasEmail> ?vcard_email .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Email> .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Work> .
?vcard_email <http://www.w3.org/2006/vcard/ns#email> ?email .
}

WHERE {

?person <http://localhost/vivo/ws_ppl_name[71]> ?fullname .

OPTIONAL { ?person <http://localhost/vivo/ws_ppl_first[72]> ?first . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_middle[73]> ?middle . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_last[74]> ?last . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_email[75]> ?email . }
?person <http://localhost/vivo/ws_ppl_person_ID[76]> ?hrid .
BIND(URI(CONCAT(STR( ?person ), "_vcard")) AS ?vcard ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardname")) AS ?vcard_name ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardemail")) AS ?vcard_email ) .

}
```

Next:

- Select Source Model ('csv-ingest')
- Select Destination Model ('csv-construct')
- Select "Execute CONSTRUCT"
- Upon completion, the system will report 'n statements CONSTRUCTed'.

Note, the BIND statements in the query allow us to create unique URIs for the associated vCard objects required in VIVO v1.6+.

7.1.1.2.7 *Step 6*: Load to Webapp

The live webapp that is indexed does not allow for models to just be attached. Attaching models works well for seeing what we have constructed or ingested or smushed, but those models are lost when Tomcat refreshes.

The final step is to output the final model and add it into the current model

1. From the Ingest Menu, select "Manage Jena Models"
2. Click "output model" below 'csv-construct'

---

[71] http://vivo.coop-plus.eu/ws_ppl_name
[72] http://vivo.coop-plus.eu/ws_ppl_first
[73] http://vivo.coop-plus.eu/ws_ppl_middle
[74] http://vivo.coop-plus.eu/ws_ppl_last
[75] http://vivo.coop-plus.eu/ws_ppl_email
[76] http://vivo.coop-plus.eu/ws_ppl_person_ID

3. Save the resulting file
4. Navigate back to the Site Administration page
5. Select Add/Remove RDF
6. Browse to the file previously saved
7. Select N3 as import format*
8. Confirmation should state ‘Added RDF from file people_rdf. Added 415 statements.’
    - The output engine uses N3, not RDF/XML. This is important to note when adding the ‘output mode’ RDF data into the webapp. RDF/XML is the default setting for the drop down list as most ontologies are written in RDF/XML.

The ingested data should now display in both the index and the search results. It is part of the main webapp and will be retained upon a Tomcat restart. The founding steps of data ingest have been completed. Repeat steps 4 and 5 for organizations and people using the SPARQL queries supplied in the appendix below.

7.1.1.3 **Appendix A: SPARQL Queries**

7.1.1.3.1 People Construct:

```
CONSTRUCT {

?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#FacultyMember> .
?person <http://www.w3.org/2000/01/rdf-schema#label> ?fullname .

?person <http://purl.obolibrary.org/obo/ARG_2000028> ?vcard .
?vcard <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Individual> .
?vcard <http://www.w3.org/2006/vcard/ns#hasName> ?vcard_name .
?vcard_name <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Name> .

?vcard_name <http://www.w3.org/2006/vcard/ns#givenName> ?first .
?vcard_name <http://vivoweb.org/ontology/core#middleName> ?middle .
?vcard_name <http://www.w3.org/2006/vcard/ns#familyName> ?last .

?vcard <http://www.w3.org/2006/vcard/ns#hasEmail> ?vcard_email .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Email> .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Work> .
?vcard_email <http://www.w3.org/2006/vcard/ns#email> ?email .
}

WHERE {
```

```
?person <http://localhost/vivo/ws_ppl_name[77]> ?fullname .

OPTIONAL { ?person <http://localhost/vivo/ws_ppl_first[78]> ?first . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_middle[79]> ?middle . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_last[80]> ?last . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_email[81]> ?email . }
?person <http://localhost/vivo/ws_ppl_person_ID[82]> ?hrid .
BIND(URI(CONCAT(STR( ?person ), "_vcard")) AS ?vcard ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardname")) AS ?vcard_name ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardemail")) AS ?vcard_email ) .

}
```

7.1.1.3.2 Organization Construct:

```
CONSTRUCT {

?org <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type_uri .
?org <http://localhost/vivo/ontology/vivo-local#orgID[83]> ?deptID .
?org <http://www.w3.org/2000/01/rdf-schema#label> ?name .
}


WHERE {
?org <http://localhost/vivo/ws_org_org_ID[84]> ?deptID .
?org <http://localhost/vivo/ws_org_org_name[85]> ?name .
?org <http://localhost/vivo/ws_org_org_vivo_uri[86]> ?type .

BIND(URI(?type) as ?type_uri)
}
```

7.1.1.3.3 Basic Position to People & Organization Construct:

Note: The example query does not account for start dates

---

[77] http://vivo.coop-plus.eu/ws_ppl_name
[78] http://vivo.coop-plus.eu/ws_ppl_first
[79] http://vivo.coop-plus.eu/ws_ppl_middle
[80] http://vivo.coop-plus.eu/ws_ppl_last
[81] http://vivo.coop-plus.eu/ws_ppl_email
[82] http://vivo.coop-plus.eu/ws_ppl_person_ID
[83] http://vivo.coop-plus.eu/ontology/vivo-local#orgID
[84] http://vivo.coop-plus.eu/ws_org_org_ID
[85] http://vivo.coop-plus.eu/ws_org_org_name
[86] http://vivo.coop-plus.eu/ws_org_org_vivo_uri

```
CONSTRUCT {

?position <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/
core#FacultyPosition> .
?position <http://www.w3.org/2000/01/rdf-schema#label> ?poslabel .
?position <http://vivoweb.org/ontology/core#relates> ?org .
?position <http://vivoweb.org/ontology/core#relates> ?person .

}
WHERE {

?org <http://localhost/vivo/ws_org_org_ID[87]> ?deptID .
?org <http://localhost/vivo/ws_org_org_name[88]> ?name .
?position <http://localhost/vivo/ws_post_department_ID[89]> ?postOrgID .
?org <http://localhost/vivo/ws_org_org_ID[90]> ?orgID .
FILTER((?postOrgID)=(?orgID))


?position <http://localhost/vivo/ws_post_department_ID[91]> ?orgID .
?position <http://localhost/vivo/ws_post_person_ID[92]> ?posthrid .
?position <http://localhost/vivo/ws_post_job_title> ?poslabel .
?person <http://localhost/vivo/ws_ppl_person_ID[93]> ?perhrid .
FILTER((?posthrid)=(?perhrid))

}
```

7.1.2 Data types for string and language

VIVO 1.10 implements RDF 1.1 and Jena 3. These changes impact the datatypes for strings and the use of the lang tag to indicate the language of the string. Please read these recommendations carefully. Jena2tools and Jena3tools will convert from previous representations to the representations recommended here.

7.1.2.1 Literal Values

Jena 3 improves Jena's RDF 1.1 compatibility. Specifically, literal values are always stored internally with datatypes. "Untyped" string literals are the same as the identical value typed as xsd:string. See the following document for more information https://jena.apache.org/documentation/migrate_jena2_jena3.html

---

[87] http://vivo.coop-plus.eu/ws_org_org_ID
[88] http://vivo.coop-plus.eu/ws_org_org_name
[89] http://vivo.coop-plus.eu/ws_post_department_ID
[90] http://vivo.coop-plus.eu/ws_org_org_ID
[91] http://vivo.coop-plus.eu/ws_post_department_ID
[92] http://vivo.coop-plus.eu/ws_post_person_ID
[93] http://vivo.coop-plus.eu/ws_ppl_person_ID

For VIVO, this means that the two triples:

```
<subj> <pred> "value"
<subj> <pred> "value"^^xsd:string
```

will be treated as the same triple and only stored once in your triple store. As a result, when using the procedure described below to upgrade your triple store from an earlier version of VIVO, you may find that the number of triples in your triple store after the upgrade is lower than the number before the upgrade.

Any code, tools, parsers, utilities, or queries that expect to differentiate between these two triples will produce results different than produced previously – RDF 1.1 no longer distinguishes between these two triples. In particular, queries that limit results based on the xsd:string datatype will likely produce larger result sets, as previously untyped triples are now typed as xsd:string internally.

Another way of saying this – any triple loaded into VIVO or Vitro that does not have a type will be typed as xsd:string internally.

Exports from VIVO and Vitro will never include the xsd:string datatype. Literal values that do not have explicit types are always assumed to be xsd:string.

As a result, we recommend that input process for VIVO do not include xsd:string datatypes on literals. While they may be correct, and will result in the literal value being typed as xsd:string internally, export processes will not include the xsd:string on output.

In RDF 1.0, a type could not be asserted with a language identifier. In RDF 1.1, a type can be asserted with a language identifier. Untyped input with language identifiers were left as untyped internally in RDF 1.0. In RDF 1.1, untyped input with language identifiers are assumed to have type rdf:langString. Exports from VIVO for triples with language tags will never include the rdf:langString datatype. Literal values with language tags are always assumed to be rdf:langString.

As a result, we recommend that input process for VIVO do not include datatypes on triples with language types. While asserting rdf:langString is correct, and will result in the literal value being typed as rdf:langString internally, export processes will not include the rdf:langString on output.

Code, tools, parsers, utilities based on RDF 1.0 should not be used with Vitro and VIVO 1.10.x All code, tools, parsers, and utilities must support RDF 1.1.

On start-up of version 1.10.x, the triple store is checked to insure that it has been upgraded. If untyped literals are found in the triple store, an error message will appear in the browser and the application will not start. The test applies only to the content store. It is possible that your content store could pass this test, but your configuration triple store remains incompatible with Jena 3 and RDF 1.1. In such a case, your application may become unstable.

7.1.2.2 Recommendations

1. String literals should be untyped in RDF input to VIVO. Use "xxx" rather than "xxx"^^xsd:string
2. Lang tags should be used with untyped string literals in RDF input to VIVO. Use "xxx"@fr-CA rather than "xxx"@fr-CA^^rdf:langString
3. Lang tags should be used on all strings which might render differently in different languages. Use "United States"@en-US. For strings which are not rendered differently in different languages use a simple untyped string literal. For example "0000-0001-2345-321X"

7.2 Exporting Data from VIVO

7.2.1 Exporting All Data

To export all data data from VIVO, use jena3tools, a command line utility provided with VIVO. jena3tools can export the VIVO content and configuration into files in a format of your choosing. See jena3tools (see page 503) for more information.

7.2.2 Exporting Selected Data

To export selected data from VIVO, use a SPARQL query. See SPARQL Queries (see page 118). SPARQL can export data in a variety of formats, including CSV, JSON, and RDF/XML.

If you need to export selected data repeatedly, say to provide a list of recent publications, or to provide data for institutional web sites, you may wish to provide an API endpoint that you configure to provide data, including parameterized data. See the Data Distribution API (see page 339)

If you are writing software to access VIVO, there are multiple APIs that can be used. See APIs (see page 338). In particular, you may wish to consider Triple Pattern Fragments (see page 380), which provides an easy and very fast mechanism for retrieving data from VIVO.

7.3 Managing Person Identifiers

VIVO provides several means for specifying various identifiers with people. Sign on to VIVO as an editor and open the Identity tab. You will see various identifiers supported by VIVO.

The table below lists the identity options provided by VIVO. Additional identifiers can be added by Extending the VIVO ontology[94]. See the Notes below the table for additional information regarding identity and identifiers in VIVO.

Identifier	Description	Data or Object	Predicate	Creates External Link
sameAs	owl sameAs assertion. See http://www.w3.org/TR/owl-ref/#sameAs-def	Object	owl:sameAs	No
ORCID iD	The Open Researcher and Contributor ID See http://orcid.org	Object	vivo:orcidId	Yes

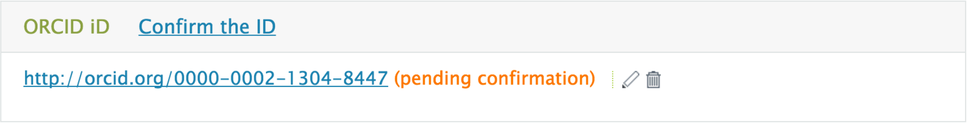
---

[94] https://wiki.lyrasis.org/display/VIVO/Extending+the+VIVO+ontology

Identifier	Description	Data or Object	Predicate	Creates External Link
eRA Commons ID	The US NIH Electronic Research Administration Commons ID. See https://commons.era.nih.gov/	Data	vivo:eRACommonsId	No
ResearcherID	Clarivate ResearcherID See http://www.researcherid.com/	Data	vivo:researcherId	No
Scopus ID	Elsevier Scopus Author Identifier. See http://help.scopus.com/Content/h_autsrch_intro.htm	Data	vivo:scopusId	Yes
Health Care Provider ID	Generic field for holding a health care provider ID of the institution's choice.	Data	obo:ARG_0000197	No

7.3.1 Notes

1. Many identifiers are configured by default to not be publicly displayed. To change the display level of an identifier, go to Site Admin > Property Groups, select the identifier in the identity section and then click Edit Property Record. Alternatively, make the change in initialTBoxAnnotations.n3.
2. sameAs can be configured as supported or unsupported in VIVO. See VIVO v1.6 release planning[95]
3. ORCID can be configured for integration with the ORCID. See Activating the ORCID integration (see page 311)
    a. To add ORCID identifiers using RDF, assert the triple associating the personURI with orcidURI: <personURI> <vivo:orcidId> <orcidURI>, where orcidURI looks like http://orcid.org/xxxx-xxxx-xxxx-xxxx
    b. Add a second triple to assert that the orcidURI is a thing: <orcidURI> a owl:Thing .
    c. When these two triples are added for a person, the VIVO interface will report the ORCID as unconfirmed.



http://orcid.org/0000-0002-1304-8447 (pending confirmation)

---

[95] https://wiki.lyrasis.org/display/VIVO/VIVO+v1.6+release+planning#VIVOv1.6releaseplanning-sameAs

   d. The user can logon to VIVO, select "Confirm the ID" and enter their ORCID password. The ORCID iD will then be confirmed in VIVO.
   e. Or you can confirm the ORCID by adding another triple to VIVO: <orcidURI> vivo:confirmedOrcidId <personURI>

4. For a SCOPUS ID, VIVO provides a link to SCOPUS at http://www.scopus.com/authid/detail.url?authorId=nnnnnnnn

7.4 Managing Organization Hierarchy

7.4.1 Overview

VIVO can be used to mange the hierarchical structure, or organizational chart, of any organization. We describe how VIVO represents organizational structure, and how to create data that can be loaded into VIVO to represent the structure of an organization. The Sample Data (see page 54) has a sample university, with sample colleges and departments, illustrating the techniques described here.

7.4.2 "hasPart, partOf"

VIVO uses the Basic Formal Ontology "hasPart" to indicate that organization A "hasPart" organization B. Sample University has a College of Science. We say College of Science "partOf" Sample University. VIVO will automatically assert Sample University "hasPart" College of Science.

The Basic Formal Ontology has been implemented as part of the Open Biomedical Ontologies (OBO). They have coded "hasPart" as "BFO_0000051" and "partOf" is "BFO_0000050" VIVO uses the OBO representation which can be found here: http://purl.obolibrary.org/obo/

7.4.3 Your Organizational Data

To manage organizations in VIVO, you will need to create data representing your organizations and your organizational structure. This is easy to do using a spreadsheet. Simply create one row for each organization in your institution. For example, the table below shows two colleges as part of Sample University, and each college has one or more departments. The Department of Chemistry has a division – Organic Chemistry.

Your URI will contain your domain name. In this example we have structured the URI. VIVO URI always start with the domain, followed by "/individual/" followed by a URI of your choice. Here we have used a tag "org" to indicate URIs for organizations. This will make the URI easier to find. We then use the name of the organization, with each word capitalized. URI must be unique. If two organizations within your institution have the same name, you will need to make unique URI for each – you might add a "1" to the end of one of the URI, for example.

Add as many rows as it takes to represent the organizations within your institution. You can start with some, load them to VIVO (see below), and create more rows and reload, eventually building up a complete set of organizations.

OrgURI	Type	Name	PartOfURI
http://vivo.mydomain.edu/individual/orgSampleUniversity	university	Sample University	
http://vivo.mydomain.edu/individual/orgCollegeOfScience	college	College of Science	http://vivo.mydomain.edu/individual/orgSampleUniversity
http://vivo.mydomain.edu/individual/orgPhysics	department	Physics	http://vivo.mydomain.edu/individual/orgCollegeOfScience
http://vivo.mydomain.edu/individual/orgChemistry	department	Chemistry	http://vivo.mydomain.edu/individual/orgCollegeOfScience
http://vivo.mydomain.edu/individual/orgOrganicChemistry	division	Organic Chemistry	http://vivo.mydomain.edu/individual/orgChemistry
http://vivo.mydomain.edu/individual/orgCollegeOfTheArts	college	College of the Arts	http://vivo.mydomain.edu/individual/orgSampleUniversity
http://vivo.mydomain.edu/individual/orgTheaterAndDance	department	Theater and Dance	http://vivo.mydomain.edu/individual/orgCollegeOfTheArt

7.4.4 Making Triples

To load data into VIVO, you will make triples out of your organizational data. You can use tools such as Karma, SAS, python, the VIVO Pump, VIVO Harvester, XSLT, your own scripts, or a text editor.

To load your organizational data into VIVO, you will transform your spreadsheet into triples. Each row of the spreadsheet will result in three triples: 1) a triple that asserts that the OrgURI is an organization of a particular type; 2) a triple that asserts that the organization has a particular name; and 3) a triple that asserts that the organization is part of another organization. Notice that the first row in the spreadsheet will generate two triples – Sample University is not part of another organization. The other six rows will generate 3 triples each. We will have 20 triples in all 2 + 3 * 6 = 20. The triples are shown below. Notes follow.

Sample Triples

```
<http://vivo.mydomain.edu/individual/orgSampleUniversity> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#University> .
<http://vivo.mydomain.edu/individual/orgSampleUniversity> <http://www.w3.org/2000/01/rdf-schema#label> "Sample University"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#College> .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://www.w3.org/2000/01/rdf-schema#label> "College of Science"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgSampleUniversity> .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Department> .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://www.w3.org/2000/01/rdf-schema#label> "Physics"@en-US .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgCollegeOfScience> .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Department> .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://www.w3.org/2000/01/rdf-schema#label> "Chemistry"@en-US .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgCollegeOfScience> .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Division> .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://www.w3.org/2000/01/rdf-schema#label> "Organic Chemistry"@en-US .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgChemistry> .
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#College> .
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://www.w3.org/2000/01/rdf-schema#label> "College of the Arts"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgSampleUniversity> .
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Department> .
```

```
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://www.w3.org/2000/01/rdf-schema#label> "Theater and Dance"@en-US .
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> .
```

7.4.4.1 Notes regarding the triples

1. The triples are shown in N-triples format[96] and should be stored in a file with the file type ".nt" Elements of the triples are either URI (in brackets) or strings (in double quotes with a language tag). Each triple ends with a period.
2. Each triple is of the form subject predicate object. The subject is always a URI (in brackets) and is a URI you specify in your table of data.
3. The predicates are from the ontologies VIVO uses to represent data. They are also always URI (in brackets). There are three kinds of assertions being made. Each has its own predicate:
    a. The predicate to assert a type is <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    b. The predicate to assert a name, known as a label, is <http://www.w3.org/2000/01/rdf-schema#label>
    c. The predicate to assert that the organization is part of another organization is <http://purl.obolibrary.org/obo/BFO_0000050>

7.4.5 Managing the Triples in VIVO

Now that we have triples representing our organizations and their organization hierarchy, its easy to load them into VIVO and update them as necessary. We will load the triples into a "named graph." This is just a collection of triples with a name. We will use a named graph to make it easy to update – we'll just empty the named graph and reload it. That way we can be sure that the data in the file of triples is the data in the named graph.

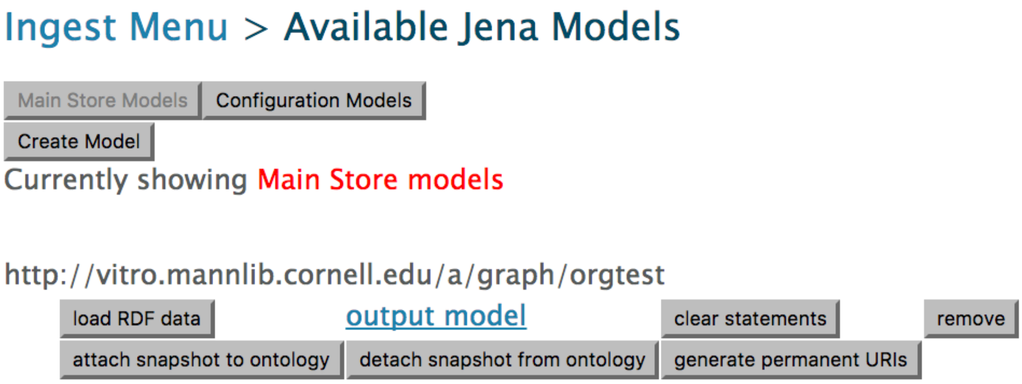
7.4.5.1 Add the triples to VIVO

1. Sign on VIVO as a system administrator.
2. Navigate to Site Admin > Advanced Data Tools > Ingest Tools > Manage Jena Models
3. At the top, click on the button "Create Model"
4. Enter the name of your model. Your name must be valid as part of a URI (no spaces). Your might call your model "orgtest"

---

[96] https://en.wikipedia.org/wiki/N-Triples

5. Vitro creates a model with your name. Find it in the list of models. You should see:



Ingest Menu > Available Jena Models

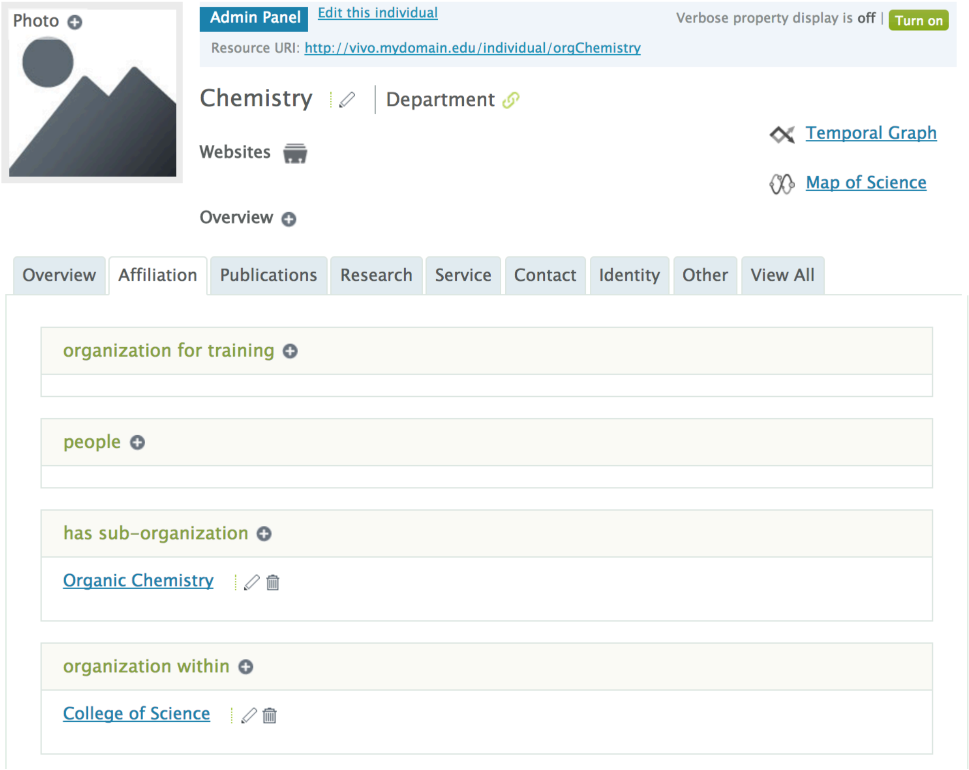
Currently showing Main Store models

http://vitro.mannlib.cornell.edu/a/graph/orgtest

Main Store Models
Configuration Models
Create Model
load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs

6. Click "load RDF data"
7. Click Choose file and select your file of triples
8. Click on the selector to indicate that your file is in "N-triples" format
9. Click Load Data

You're done! You have your organizations and your organizational hierarchy data in VIVO. You can navigate to one of your organizations and see the hierarchical information:



Resource URI: http://vivo.mydomain.edu/individual/orgChemistry

Verbose property display is off | Turn on

Chemistry | Department

has sub-organization: Organic Chemistry

organization within: College of Science

Photo
Admin Panel
Edit this individual
Websites
Overview
Temporal Graph
Map of Science
Overview
Affiliation
Publications
Research
Service
Contact
Identity
Other
View All
organization for training
people
has sub-organization
Organic Chemistry
organization within
College of Science

7.4.5.2 Updating your triples in VIVO

When you have organizational data to update – a new organization has been added, an organizational change has been made, an organization has a new name, you discovered an error in your data, or for any other reason, update the file with your triples and follow the steps below to replace your organizational data in VIVO with the organizational data in your file.

1. Sign on VIVO as a system administrator.
2. Navigate to Site Admin > Advanced Data Tools > Ingest Tools > Manage Jena Models
3. Find your organization named graph it in the list of models.
4. Carefully click "clear statements" for your organization named graph. Be careful not to clear the statements of any other models.
5. Click "Load RDF data"
6. Click Choose file and select your file of triples
7. Click on the selector to indicate that your file is in "N-triples" format
8. Click Load Data

Your previous organization data has been replaced with your new organizational data.

7.4.6 Some Closing Observations

The basic technique for data management described here – creating triples, loading them into a named graph, and updating the graph when data changes – can be used to manage any of your VIVO data. You can put people in one graph, publications in another, grants in another, datasets in another, and manage each by creating triples and updating graphs. You may wish to create repeatable processes for each of the kinds of data you are managing. These processes should be based on tools of your choice – Karma, XSLT, or scripts you write.

The Ontology Reference (see page 446) provides details regarding how VIVO represents entities. Practice will lead to familiarity and the VIVO community is always happy to help with any questions you may have.

7.5 Managing Data Packages

7.5.1 Overview

Data packages are sets of data represented using VIVO RDF in one of the supported VIVO RDF file formats – Turtle (.ttl), Triples (.nt), Notation3 (.n3) or RDF-XML (.rdf) Data packages are typically produced as semi-static – they can be loaded into VIVO and updated as needed. Data packages typically deliver statements about entities outside the management of the particular VIVO.

VIVO manages data packages by creating a new graph for each package, containing the asserted triples for the data package and named with the name of the data package file. The VIVO inferencer creates inferred triples for the data packages and stored the inferred triples in the inference graph. When changes are made to the data package file, the VIVO inferencer must be run to bring the inference graph up to date with the changes made in the asserted graph for the data package.

An example of a data package would be the Grid data, representing the research organizations of the world. This data set, maintained by Digital Science, contains more than 65,000 university, research institutes, funding agencies and other organizations involved in research across the world. The data set contains the official name and alternate names as well as abbreviations of names of the each organization, its geographic location, its type, date of founding, parent, child and affiliated organizations, as well as persistent identifiers for the organization. The data is available as a data package for VIVO at https://github.com/openvivo/grid-rdf

7.5.2 Add a data package

To add a data package to VIVO,

1. Place a copy of the data package file in `vivo/home/rdf/abox/filegraph`
2. Restart Tomcat. VIVO will add a new graph to the triple store containing the asserted triples in the data package file. See Graph Reference (see page 443) for additional detail. The VIVO inferencer will infer additional triples regarding the data package and add those triples to the vitro-kb-info graph. Again see Graph Reference (see page 443). Note: the inferencer may take quite awhile to complete. Adding a package with tens of thousands of new entities, each with dozens of attributes may take hours to reinference.

7.5.3 Update a data package

To update a data package:

1. Place a copy of the updated data package file in
```
      vivo/home/rdf/abox/filegraph
```
2. Restart Tomcat. VIVO will compare the contents of the triple store with the contents of the data package file, and update the triples in the associate graph as needed. VIVO will then reinference the triple store. Note: the inferencer may take quite awhile to complete.

7.5.4 Delete a data package

To delete a data package:

1. Remove the data package file from `vivo/home/rdf/abox/filegraph`
2. Restart Tomcat. VIVO will detect that the file is no longer present and remove the associated graph. The inferencer will be run and triples in the inference graph associated with the deleted data package file will be removed from the inference graph.

7.5.5 Available Data Packages

Data packages are available at the following locations:

1. Research organizations of the world. From http://gid.ac Available as CC-0 data. https://github.com/openvivo/grid-rdf
2. Journals of the world. Compiled from CrossRef and NIH PubMed. More than 40K journals, each with title and ISSN. https://github.com/OpenVIVO/OpenVIVOjournals
3. Dates. Dates with simple URI, known URI. Avoid creating multiple date entities for the same date. Link all references to a date to a single date entity. https://github.com/OpenVIVO/date-rdf
4. Cities of the United States. Data for all cities in the United States with population 100K or more. Includes lat/long. https://github.com/mconlon17/vivo-add-cities

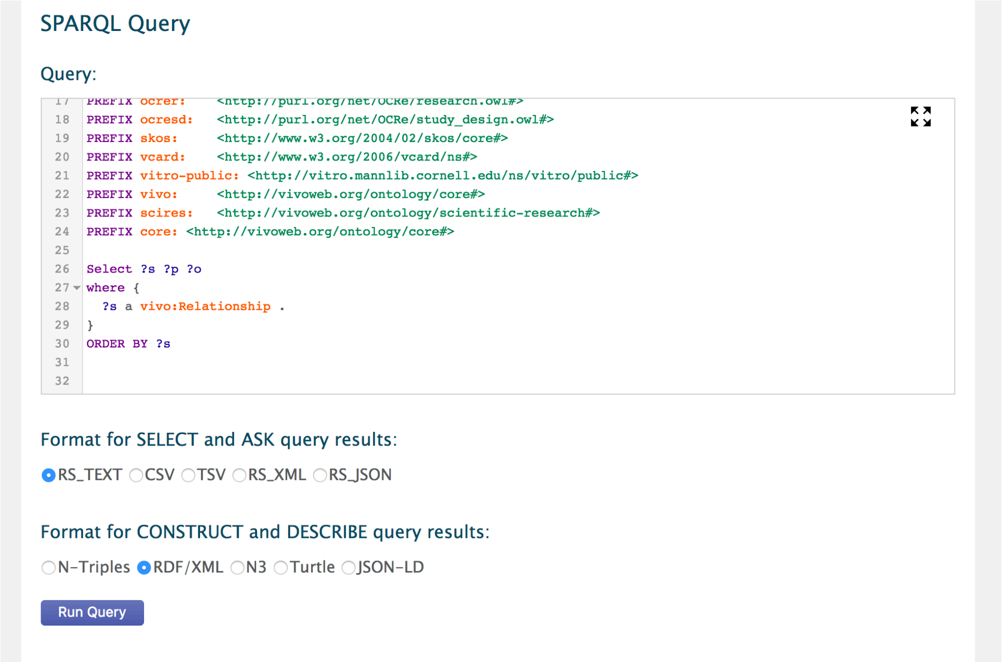
7.6 SPARQL Queries

7.6.1 Overview

SPARQL is a query language for RDF-based systems such as Vitro and VIVO. Using SPARQL one can extract any information from VIVO or Vitro, producing reports, or providing data for other software such as visualizations or user interfaces.

7.6.2 Running SPARQL queries

To run a SPARQL query, navigate to Site Admin > Sparql query. You will see a page similar to the one below. Note that the text is in various colors. VIVO uses the YASQE editor for SPARQL. You can read more about its features at their web site. See http://yasqe.yasgui.org/ At the top of the Query window are SPARQL prefix declarations for VIVO. These provides abbreviations for various URLs you use in your SPARQL queries. For more on SPARQL, see Learning SPARQL. http://learningsparql.com   Below the prefix declarations is the actual SPARQL query.



SPARQL Query

Query:

```
PREFIX ocrer:    <http://purl.org/net/OCRe/research.owl#>
PREFIX ocresd:   <http://purl.org/net/OCRe/study_design.owl#>
PREFIX skos:     <http://www.w3.org/2004/02/skos/core#>
PREFIX vcard:    <http://www.w3.org/2006/vcard/ns#>
PREFIX vitro-public: <http://vitro.mannlib.cornell.edu/ns/vitro/public#>
PREFIX vivo:     <http://vivoweb.org/ontology/core#>
PREFIX scires:   <http://vivoweb.org/ontology/scientific-research#>
PREFIX core: <http://vivoweb.org/ontology/core#>

Select ?s ?p ?o
where {
  ?s a vivo:Relationship .
}
ORDER BY ?s


```

Format for SELECT and ASK query results:

● RS_TEXT ○ CSV ○ TSV ○ RS_XML ○ RS_JSON

Format for CONSTRUCT and DESCRIBE query results:

○ N-Triples ● RDF/XML ○ N3 ○ Turtle ○ JSON-LD

The query in the figure above asks for a sorted list of the entities in VIVO that have type Relationship. RS_TEXT has been selected as an output format. This will display in a browser window. Note that you can select CSV, or TSV. These will download to your computer. You may also select RS_XML and RS_JSON – these will also display in your browser window.

Running the above query in OpenVIVO generates over 10,000 rows of output. The beginning of the output in JSON format is shown below:

First few lines of JSON output for preceding query

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship2" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/m9.figshare.2002020-authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/
m9.figshare.2002200-authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/
m9.figshare.2002200-authorship2" }
      } ,
```

7.6.3 Using SPARQL for reporting

SPRQL can be used to extract data from VIVO for reporting. Using the TSV (tab separated values) output format, the results of a SPARQL query can be downloaded from VIVO and uploaded to a spreadsheet, reporting or presentation tool.

The query below makes a contact list for all people in particular academic unit.

```
#
# Find all the people with a position in the CTSI or any CTSI sub-unit,
# and list them alphabetically with phone, email, gatorlink, eracommons if any
#
SELECT ?person (MIN(DISTINCT ?xname) AS ?name)
    (MIN(DISTINCT ?xphone) AS ?phone)
    (MIN(DISTINCT ?xemail) AS ?email)
    (MIN(DISTINCT ?xgatorlink) AS ?gatorlink)
    (MIN(DISTINCT ?xeracommons) AS ?eracommons)
WHERE {
  {?pos vivo:relates <http://vivo.ufl.edu/individual/n8763427> . ?pos a
vivo:Position .}
  UNION
  {<http://vivo.ufl.edu/individual/n8763427> obo:BFO_0000051 ?sub .
   ?pos vivo:relates ?sub . ?pos a vivo:Position .}
  ?pos vivo:dateTimeInterval ?dt .
  OPTIONAL {?dt vivo:end ?end . }
  FILTER (!BOUND(?end))  # current positions do not have end dates
  ?pos vivo:relates ?person . ?person a foaf:Person .
  ?person rdfs:label ?xname .
  ?person a ufVivo:UFCurrentEntity .
  ?person obo:ARG_2000028 ?vcard .
  OPTIONAL { ?vcard vcard:hasEmail ?email_thing . ?email_thing vcard:email  ?
xemail .}
  OPTIONAL { ?vcard vcard:hasTelephone ?tel_thing . ?tel_thing vcard:telephone ?
xphone .}
  OPTIONAL { ?person ufVivo:gatorlink ?xgatorlink .}
  OPTIONAL { ?person vivo:eRACommonsId ?xeracommons .}
}
GROUP BY ?person
ORDER BY ?name
```

For additional examples, some much simpler than the example above, see @Mike Conlon's web site http://mconlon17.github.io/sparql

7.6.4 Using SPARQL to clean data

Using SPARQL queries, one can find triples meeting a criteria for improvement. You might query for people without labels, for example.

CONSTRUCT statements can be used to make triples that you would like to remove from your VIVO. Run the query to make the triples, download them, then use System Admin > Add/Remove RDF data to Remove the triples you have constructed. In a similar manner, you can construct improved triples and add them to you VIVO.

7.6.5 DESCRIBE queries

Vitro SPARQL supports DESCRIBE queries, but DESCRIBE is not well-defined by W3C standards, allowing implementation specific variations. In Vitro, a request to DESCRIBE a URL will return all the triples with that URL as the subject. So, for example:

A sample DESCRIBE query

```
DESCRIBE <http://openvivo.org/a/doi10.4225/03/58ca600d726bd>
```

returns

```
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.obolibrary.org/obo/
ARG_2000028> <http://openvivo.org/a/doi10.4225/03/58ca600d726bd-vcard> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#relatedBy> <http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship2> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/BFO_0000031> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.obolibrary.org/obo/
RO_0002353> <http://openvivo.org/a/eventVIVO2017> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/
abstract> "<div>This is the pre-print version of a paper accepted in Open Repository
Conference in Brisbane, Australia, June 2017.<b><br></b></div><div><br></
div><b>Abstract\u00A0</b><div><b><br></b><div>Research Graph is an open collaborative
project that builds the capability for connecting researchers, publications, research
grants and research datasets (data in research). \u00A0VIVO is an open source,
semantic web platform and a set of ontologies for representing scholarship. \u00A0To
provide interoperability between Research Graph data and VIVO systems we modelled the
Research Graph metamodel using the VIVO Integrated Semantic Framework. To evaluate
the mapping, we used the model to connect figshare RDF records to data collections in
Research Data Australia using Research Graph API. In addition, we are working toward
loading Research Graph data into a VIVO instance. \u00A0VIVO provides a search
capability, and pages for first class entities in the Research Graph model --
researcher, dataset, grant, and publication. \u00A0The result provides a
visualisation solution for co-authors, co-funding, timeline, and a capability map for
finding expertise related to concepts of interest. \u00A0The resulting linked open
```

```
data will be made freely available and can be used in other tools for additional
discovery.<br></div></div>" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/
doi> "10.4225/03/58ca600d726bd" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#datePublished> <http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#relatedBy> <http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship1> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/
freetextKeyword> "Research Discovery" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Thing> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#dateCreated> <http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.org/ontology/bibo/Document> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/
freetextKeyword> "Linked Open Data" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://
vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> <http://vivoweb.org/
ontology/core#ConferencePaper> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/BFO_0000001> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/2000/01/rdf-
schema#label> "Creating an open linked data model for Research Graph using VIVO
Ontology" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/
freetextKeyword> "Open research outputs" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/IAO_0000030> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/BFO_0000002> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#dateTimeValue> <http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/
core#dateModified> <http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#ConferencePaper> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://purl.org/ontology/bibo/Article> .
```

7.6.6 ASK Queries

Vitro SPARQL supports ASK queries which return either true (there are triples that satisfy the pattern), or false (there are no triples that satisfy the pattern). The query below will return true in most VIVOs and false in a new VIVO.

ASK Query

```
ASK { ?s a vivo:Relationship . }
```

7.6.7 Additional SPARQL Resources

- YASQUE, "YASQUE Home Page," web site. Last Accessed June 17, 2017. http://yasqe.yasgui.org/
- Conlon, M. "Sample SPARQL: SPARQL scripts for getting information from your VIVO," web site. Last Accessed June 17, 2017. http://mconlon17.github.io/sparql/
- DuCharme, B. "Learning SPARQL," Wiley Publishing, 2011. 235 pages. http://www.learningsparql.com/
- Apache Jena, "SPARQL Tutorial," web site. Last accessed June 17, 2017. https://jena.apache.org/tutorials/sparql.html

7.7 How to remove data from a specific graph

Using the Ingest Tools,

1. Go to Manage Jena Models and add a new, temporary model.
2. Use the "load RDF data" button below it to add to this temporary model the RDF you ultimately want to delete.
3. Go to back to the Ingest Tools menu, select Subtract One Model from Another.
4. Set "model to be subtracted from" and "model in which difference should be saved" to the graph you wanted to delete from in the first place. Set "model to subtract" to the temporary graph you just created.
5. Run the subtraction.
6. Go back to Manage Jena Models and remove the temporary model.

7.8 Removing Entities from VIVO

7.8.1 General Method

To remove entities from VIVO, run SPARQL queries to retrieve the triples for the entities as RDF. Then go to Site Administration -> Advanced Data Tools -> Add or Remove RDF Data to upload the RDF to remove the triples for the entities.

To make this functionality more user-friendly and preserve usage of construct query for deletion of linked instances for the certain type of entity, a display object property display:hasDeleteQuery linked with that entity and with the construct query should be written in a file located in the VIVO_HOME directory rdf/display/everytime/. To display the option for deletion of an instance and linked instances, add link

<@p.deleteIndividualLink individual /> to an individual page. There is already a trash bin icon in profiles (created by <@p.deleteIndividualLink individual />) which is leading to individual deletion page.

Entities that are involved in relationships will need more attention. The relationship involving the entity should also be removed.

7.8.2 Examples

7.8.2.1 Remove publications by type

Run the following SPARQL CONSTRUCT queries to retrieve the triples associated with the entities:

7.8.2.1.1 Article

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type bibo:Article .
     ?s ?p ?o  .
}
```

7.8.2.1.2 Book

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type bibo:Book .
     ?s ?p ?o  .
}
```

7.8.2.1.3 Case Study

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type vivo:CaseStudy .
     ?s ?p ?o  .
}
```

7.8.2.1.4 Conference Paper

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type vivo:ConferencePaper .
     ?s ?p ?o  .
}
```

7.8.2.1.5 Editorial Article

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type vivo:EditorialArticle .
     ?s ?p ?o  .
}
```

7.8.2.1.6 Proceedings

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type bibo:Proceedings .
     ?s ?p ?o  .
}
```

7.8.2.1.7 Review

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type vivo:Review .
     ?s ?p ?o  .
}
```

7.8.2.1.8 Academic Article

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type bibo:AcademicArticle .
     ?s ?p ?o  .
}
```

7.8.2.2 Remove Other Entities

7.8.2.2.1 Journal

```
construct {
        ?s ?p ?o .
} where {
     ?s rdf:type bibo:Journal .
     ?s ?p ?o  .
}
```

7.9 Uploading files associated with individuals

7.9.1 Configuration of the runtime.properties

Before you run the Tomcat server with VIVO, please configure the following in {VIVO_HOME}/config/runtime.properties:

```
# File upload file size in bytes. By default 10485760 bytes (10Mb)
fileUpload.maxFileSize = 10485760
#comma separated list of mime types allowed for upload
fileUpload.allowedMIMETypes = image/png, application/pdf
```

The maxFileSize property might have value between 0 and 50 Mb, while default value is 10Mb.

7.9.2 Adding support for assigning file to the certain individuals class

For instance, you can set domain for vitro-public:storedFile object property to http://purl.org/ontology/bibo/Book. After that, any instance of this class will have options to link files of the MIME type specified in the runtime.properties. The user with edit privileges can attach files and remove attached files, while the user with read privileges can download files.

8 Extending and Localizing VIVO

8.1 Overview

VIVO, and Vitro, its underlying technology, are open and flexible. There is significant opportunity to extend VIVO and/or Vitro to accommodate the needs of your institution. No extensions are needed – VIVO contains a comprehensive information representation for scholarship. Vitro provides a general purpose platform for semantic data management.

Some of the topics in this section are very common – most sites want to localize their branding, many sites use external authentication, and many sites use VIVO in languages other than english. Other topics are more advanced and less common – creating custom editing forms, for example.

Take what you need and leave the rest.

- Internationalization (see page 129)
    - Enabling Interface Languages in VIVO as an Administrator (see page 140)
    - Using VIVO's Internationalization (i18n) Features (see page 141)
    - Vitro UI labels vocabulary (see page 148)
- Customizing the Interface (see page 150)
    - Home page customizations (see page 154)
    - Menu and page management (see page 163)
    - Annotations on the ontology (see page 166)
    - Class-specific templates for profile pages (see page 183)
    - Excluding Classes from the Search (see page 188)
    - Custom List View Configurations (see page 188)
    - Creating short views of individuals (see page 196)
    - Creating a custom theme (see page 210)
    - Creating custom entry forms (see page 217)
    - Enhancing Freemarker templates with DataGetters (see page 232)
    - Enriching profile pages using SPARQL query DataGetters (see page 235)
    - Multiple profile types for foaf:Person (see page 240)
    - Using OpenSocial Gadgets (see page 245)
    - How VIVO creates a page (see page 248)
    - Tips for Interface Developers (see page 259)
- Adding Additional Ontologies to VIVO (see page 262)
- Enable an external authentication system (see page 265)
    - How User Accounts are Associated with Profile Pages (see page 265)
    - Shibboleth example (see page 266)

    - Using a Tomcat Realm for external authentication (see page 269)
- Authorization (see page 270)
    - Writing a controller for a secured page (see page 270)
    - Creating a VIVO authorization policy - an example (see page 275)
    - A more elaborate authorization policy (see page 283)
    - The IdentifierBundle - who is requesting authorization? (see page 289)
- Adding External Vocabularies (see page 292)
- Search Engine Optimization (SEO) (see page 292)

8.2 Internationalization

8.2.1 Children Pages

- Enabling Interface Languages in VIVO as an Administrator (see page 140)
- Using VIVO's Internationalization (i18n) Features (see page 141)
- Vitro UI labels vocabulary (see page 148)

8.2.2 Summary of this Page

8.2.3 VIVO Language Support

When a VIVO site supports a language other than English, that support includes:

- Text that is displayed in the VIVO pages. For example, menus, selections, prompts, tool-tips and plain text.
- Text from terms in the Ontology, which are frequently displayed as links or section headings. Text includes labels and annotations of properties and classes.
- Text values stored in the data. For example, if a book title is available in both French and English, a French-speaking user sees the French title. If a title is available only in English, it is displayed, without regard to the user's preference in languages.

Languages can be selected in a variety of ways, depending on the installation parameters:

- A VIVO administrator can configure VIVO to use one of the supported languages.
- Different users may see different languages, depending on the settings in their web browser.
- Different users may select a language from a list of available languages.

VIVO language files are available for English (U.S. and Canadian), Spanish, Brazilian Portuguese, French (Canadian) and German. If you need support for another language, please inquire of the VIVO mailing lists, to see if another group has the files you need.

8.2.4 Adding an existing language to your VIVO site

In this step by step guide we will use the German language files as an example. Be sure to use the theme 'wilma' or 'tenderfoot' for this to work without issues.

- Edit the `vivo_home_dir/config/runtime.properties` file in your VIVO home directory:
  - uncomment/add `RDFService.languageFilter = true`
  - uncomment/add `languages.selectableLocales = en_US, de_DE`
- Restart the tomcat
- You should now be able to select your installed language (in this case German) in the header of your VIVO site

For more details, see Enabling Interface Languages in VIVO as an Administrator (see page 140).

8.2.5 Building VIVO and Vitro language repositories from source (for developers)

- Clone the VIVO[97] and Vitro[98] repositories to your local machine.
- Build VIVO from the VIVO project folder using `mvn install -o -s installer/my-settings.xml` (Note the -o flag, this forces Maven to use the language projects from your local repository instead of downloading from a remote repository)
- Edit the `vivo_home_dir/config/runtime.properties` file in your VIVO home directory:
  - uncomment/add `RDFService.languageFilter = true`
  - uncomment/add `languages.selectableLocales = en_US, de_DE`
- Restart the tomcat
- You should now be able to select your installed language (in this case German) in the header of your VIVO site

8.2.6 Creating new language files for your language

First, contact the VIVO development team[99]. We would love to talk to you. We will be happy to help with any questions you may have and introduce you to others who may be working on the same language as you are.

When your files are ready, you can make them available to the development team in any way you choose. Note that the VIVO project will release your files under the Apache 2 License[100]. They will require a Contributor Agreement stating that you agree to the terms in the agreement.

Translating VIVO into your language involves determining a locale, and preparing files as discussed below.

---

[97] https://github.com/vivo-project/VIVO

[98] https://github.com/vivo-project/Vitro

[99] https://wiki.lyrasis.org/display/VIVO/Development+Interest+Group

[100] http://www.apache.org/licenses/LICENSE-2.0.html

8.2.6.1 The locale

Your locale is an internationally recognized code that specifies the language you choose, and the region where it is spoken. For example, the locale string `fr_CA` is used for French as spoken in Canada, and `es_MX` is used for Spanish as spoken in Mexico. Recognized codes for languages and regions can be found by a simple Google search. Here is a list of locales that are recognized by the Java programming language[101]. You may also use this definitive list of languages and regions[102], maintained by the Internet Assigned Numbers Authority.

The locale code will appear in the name of each file that you create. In the files that contain RDF data, the locale code will also appear at the end of each line.

> When the locale code appears in file names, it contains an underscore `(en_US`). When it appears inside RDF data files, it contains a hyphen (`en-US`).

8.2.6.2 The language files

You can get the US English (home/src/main/resources/rdf/i18n/en_US) files from the VIVO and Vitro among the vivo-project repositories (https://github.com/vivo-project), to use as a template for your own files.

The process simply consists of making a directory inside VIVO and Vitro directories home/src/main/resources/rdf/i18n/, for instance et_EE (for Estonian), and copying there the complete file hierarchy from home/src/main/resources/rdf/i18n/en_US. In all copied ttl, n3, nt files, the @en-US language tags should be replaced with @ee-EE, and associated labels should be translated from English to Estonian.

In the process of initializing the files for a new language. You will encounter the following types of file:

User interface labels

These files contain about 1500 words and phrases that appear in the VIVO/Vitro web pages.

These words and phrases have been removed from the page templates, so no programming knowledge is required to translate them.

They appear at different level in the application (the <locale> might be for instance en_US) :

- in Vitro - home/src/main/resources/rdf/i18n/<locale>/i[103]nterface-i18n/firsttime/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/<locale>/i[104]nterface-i18n/firsttime/vivo_UiLabel.ttl
    - There are also files for themes:

---

[101] https://www.oracle.com/java/technologies/javase/jdk11-suported-locales.html
[102] http://www.iana.org/assignments/language-subtag-registry
[103] https://github.com/vivo-project/VIVO/tree/i18n-redesign/home/src/main/resources/rdf/i18n/en_US/interface-i18n
[104] https://github.com/vivo-project/VIVO/tree/i18n-redesign/home/src/main/resources/rdf/i18n/en_US/interface-i18n

- home/src/main/resources/rdf/i18n/<locale>/i[105]nterface-i18n/firsttime/vivo_UiLabel_wilma.ttl and
- home/src/main/resources/rdf/i18n/<locale>/i[106]nterface-i18n/firsttime/vivo_UiLabel_tenderfoot.ttl

The structure of those files is defined in the Vitro UI label vocabulary (in Vitro - home/src/main/resources/rdf/tbox/firsttime/UILabelsVocabulary.ttl).

The application will look for an entry starting with the activated theme (like tenderfoot or wilma), then VIVO and lastly Vitro. Of course, the selected locale (UI language) will be taken into account.

NOTE: VIVO/Vitro 1.13 was based on property files located in VIVO-languages and Vitro-languages repositories. Although, those repositories have been archived, and property files have been replaced with ttl files and moved to VIVO and Vitro home directory in VIVO/Vitro 1.14.0 release, due to back compatibility VIVO/Vitro 1.14 also supports using property files. It means, instead of previously listed files, customers can add vivo_all_<locale>.property and vitro_all_<locale>.property into the webapp/src/main/webapp/i18n directory.

8.2.6.2.1 RDF data (.n3, .nt)

Data in the RDF models include labels for the properties and classes, labels for property groups and class groups, labels for menu pages and more. Here is the list of directories where one will have to create required rdf files:

- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/applicationMetadata/firsttime
- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/display/firsttime
- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/tbox/firsttime
- [Vitro]/home/src/main/resources/rdf/i18n/<locale>/display/firsttime
- [Vitro]/home/src/main/resources/rdf/i18n/<locale>/tbox/firsttime

In each case, the delivered file in English has a corresponding file with the same name but in a different directory structure (locale tag is different). for instance:

File names (Estonian)

```
[VIVO]/home/src/main/resources/rdf/i18n/et_EE/applicationMetadata/firsttime/
classgroups_labels.n3
```

In each file, labels specify text to be used by VIVO. Each label should be translated and affixed with the appropriate locale tag. See below:

Some classgroups_labels (Estonian)

```
<http://vivoweb.org/ontology#vitroClassGrouppeople>
```

---

[105] https://github.com/vivo-project/VIVO/tree/i18n-redesign/home/src/main/resources/rdf/i18n/en_US/interface-i18n
[106] https://github.com/vivo-project/VIVO/tree/i18n-redesign/home/src/main/resources/rdf/i18n/en_US/interface-i18n

```
    <http://www.w3.org/2000/01/rdf-schema#label> "inimesed"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouppublications>
    <http://www.w3.org/2000/01/rdf-schema#label> "teadus"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouporganizations>
    <http://www.w3.org/2000/01/rdf-schema#label> "organisatsioonid"@et-EE .
<http://vivoweb.org/ontology#vitroClassGroupactivities>
    <http://www.w3.org/2000/01/rdf-schema#label> "tegevused"@et-EE .
```

8.2.7 How VIVO supports languages

8.2.7.1 Language in the data model

The usual form of language support in RDF is to include multiple labels for a single individual, each with a language specifier.

In fact, any set of triples in the data model are considered to be equivalent if they differ only in that the objects are strings with different language specifiers. If language filtering is enabled, VIVO will display the value that matches the user's preferred locale. If no value exactly matches the locale, the closest match is displayed.

Consider these triples in the data:

```
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1>
"coloring" .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1>
"colouring"@en-UK .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1>
"colorear"@es .
```

VIVO would display these values as follows:

User's preferred locale	displayed text
`en_UK`	colouring
`en_CA`	colouring
`es_MX`	colorear
`fr_FR`	coloring

8.2.7.2 Language support in VIVO pages

VIVO uses the Java language's built-in framework for Internationalization based on triplets preserved in the graph base.

"Internationalization" is frequently abbreviated as "I18n", because the word is so long that there are 18 letters between the first "I" and the last "n".

In the I18n framework, displayed text strings are not embedded in the Java classes or in the Freemarker template. Instead, each piece of text is assigned a "key" and the code will ask the framework to provide the text string that is associated with that key. The framework has access to sets of properties files, one set for each supported language, and it will use the appropriate set to get the correct strings.

For example, suppose that we have:

- The text that will appear in an HTML link, used to cancel the current operation, with the key `cancel_link`.
- The title of a page used to upload an image, with the key `upload_photo`.
- The text of a prompt message, telling users how big an image must be, with the key `minimum_image_dimensions`.

The default properties file might show the English language versions of these ttl files, like this:

Excerpt from /home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel.ttl

```
prop-data:cancel_link.Vitro
    rdf:type          owl:NamedIndividual ;
    rdf:type          prop:PropertyKey ;
    rdfs:label        "Cancel"@en-US ;
    prop:hasApp       "Vitro" ;
    prop:hasKey       "cancel_link" .

prop-data:upload_photo.Vitro
    rdf:type          owl:NamedIndividual ;
    rdf:type          prop:PropertyKey ;
    rdfs:label        "Upload a photo"@en-US ;
    prop:hasApp       "Vitro" ;
    prop:hasKey       "upload_photo" .

prop-data:minimum_image_dimensions.Vitro
    rdf:type          owl:NamedIndividual ;
    rdf:type          prop:PropertyKey ;
    rdfs:label        "Minimum image dimensions: {0} x {1} pixels"@en-US ;
    prop:hasApp       "Vitro" ;
    prop:hasKey       "minimum_image_dimensions" .
```

Notice that the actual image dimensions are not part of the text string. Instead, placeholders are used to show where the dimensions will appear when they are supplied. This allows us to specify the language-dependent parts of a message in the properties file, while waiting to specify the language-independent parts at run time.

A Spanish language properties file might show the Spanish versions of these properties in a similar manner:

Excerpt from /home/src/main/resources/rdf/i18n/es/interface-i18n/firsttime/vitro_UiLabel.ttl

```
prop-data:cancel_link.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Cancelar"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "cancel_link" .

prop-data:upload_photo.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Suba foto"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "upload_photo" .

prop-data:minimum_image_dimensions.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Dimensiones mínimas de imagen: {0} x {1} pixels"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "minimum_image_dimensions" .
```

To use these strings in Java code, start with the I18n class, and the key to the string. Supply values as needed to replace any placeholders in the message.

Using I18n strings from Java code

```java
protected String getTitle(String siteName, VitroRequest vreq) {
    return I18n.text(vreq, "upload_image_page_title");
}

private String getPrompt(HttpServletRequest req, int width, int height) {
    return I18n.text(req, "minimum_image_dimensions", width, height);
}
```

Similarly, using text strings in a Freemarker template begins with the `i18n()` method.

Using I18n strings in a Freemarker template

```
<#assign text_strings = i18n() >

<a href="../cancel" >
    ${text_strings.cancel_link}
</a>

<p class="note">
    ${text_strings.minimum_image_dimensions(width, height)}
</p>
```

Here is the appearance of the page in question, in English and in Spanish:

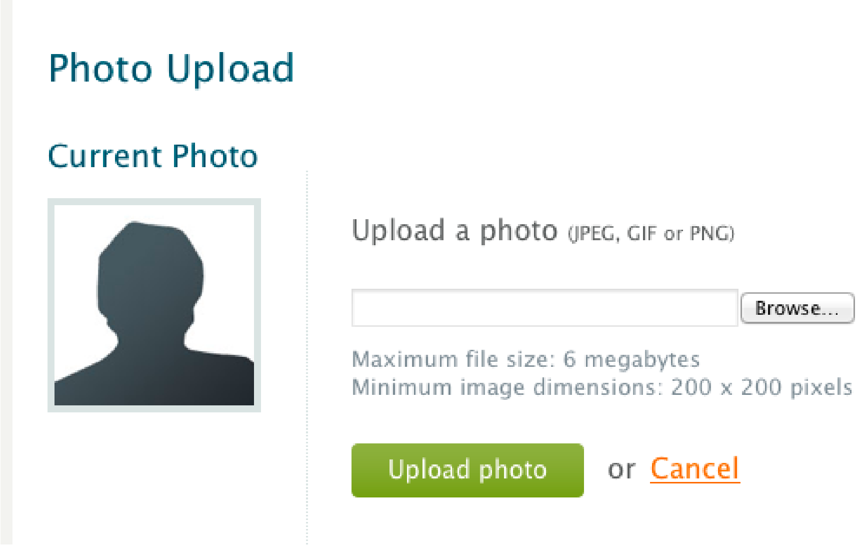
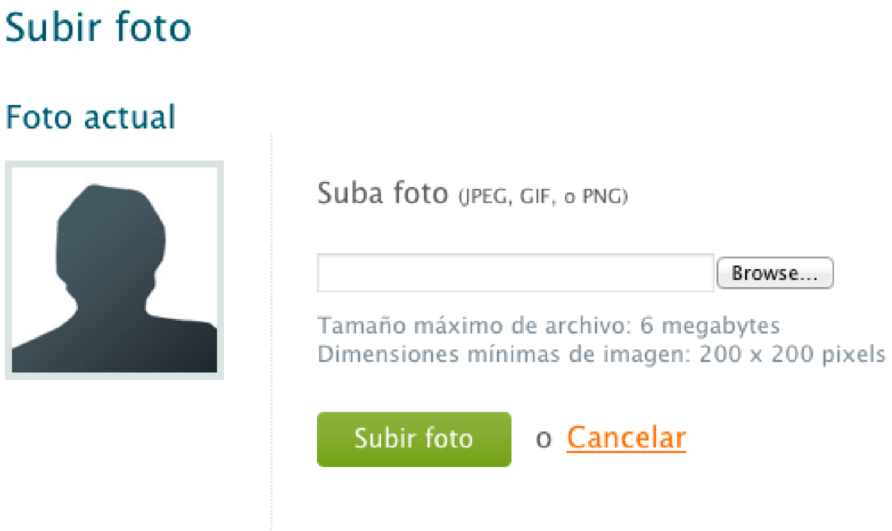


Photo Upload

Current Photo

Upload a photo (JPEG, GIF or PNG)

Maximum file size: 6 megabytes
Minimum image dimensions: 200 x 200 pixels



Subir foto

Foto actual

Suba foto (JPEG, GIF, o PNG)

Tamaño máximo de archivo: 6 megabytes
Dimensiones mínimas de imagen: 200 x 200 pixels

8.2.7.2.1 Structure of the properties files

The properties files that hold text strings are based on the Java I18n framework for resource bundles. Here is a tutorial on resource bundles[107].

Most text strings will be simple, as shown previously. However, the syntax for expressing text strings is very powerful, and can become complex. As an example, take this text string that handles both singular and plural:

A complex text string

```
prop-data:deleted_accounts.Vitro
    rdf:type          owl:NamedIndividual ;
    rdf:type          prop:PropertyKey ;
    rdfs:label        "Deleted {0} {0, choice, 0#accounts|1#account|1<accounts}."@en-
US ;
    prop:hasApp       "Vitro" ;
    prop:hasKey       "deleted_accounts" .
```

The text strings are processed by the Java I18n framework for message formats. Here is a tutorial on message formats[108]. Full details can be found in the description of the MessageFormat[109] class.

8.2.7.2.2 Local extension: application vs. theme

The Java I18n framework expects all ui labels translations to be in one location. In VIVO, this has been extended to look in three locations for text strings. First, it looks for ui label translation files in the current theme (for instance, in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_wilma.ttl). Then, it looks in the main VIVO UI labels file (in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel.ttl), and at the end in Vitro UI labels file (in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel_wilma.ttl). This means that you don't need to include all of the basic text strings in your theme. But you can still add or override strings in your theme.

If your VIVO theme is named "frodo", then your UI text strings would be in

- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel.ttl

NOTE: Please use properly hasApp and hasTheme data properties in ttl files, as well as language tags.

If you specify more than one locale for VIVO, this search pattern becomes longer. For example, if your user has chosen UQAM Canadian French as his language/country/private-use subtags combination (fr_CA_x_uqam), then these files (if they exist) will be searched for text strings:

- in VIVO - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl

---

[107] http://docs.oracle.com/javase/tutorial/i18n/resbundle/concept.html
[108] http://docs.oracle.com/javase/tutorial/i18n/format/messageintro.html
[109] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/MessageFormat.html

- in VIVO - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/**firsttime**/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/**firsttime**/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/**firsttime**/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/**firsttime**/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/**firsttime**/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr/interface-i18n/**firsttime**/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr/interface-i18n/**firsttime**/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr/interface-i18n/**firsttime**/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/**firsttime**/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/**firsttime**/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/**firsttime**/vitro_UiLabel.ttl

When VIVO finds a text string in one of these files, it uses that value, and will not search the remaining files.

8.2.7.3 Language in Freemarker page templates

Here is some example code from `page-home.ftl`

Excerpt from page-home.ftl

```
<section id="search-home" role="region">
    <h3>${i18n().intro_searchvivo} <span class="search-filter-selected">filteredSearch</span></h3>
    <fieldset>
        <legend>${i18n().search_form}</legend>
        <form id="search-homepage" action="${urls.search}" name="search-home" role="search" method="post" >
            <div id="search-home-field">
                <input type="text" name="querytext" class="search-homepage" value="" autocapitalize="off" />
                <input type="submit" value="${i18n().search_button}" class="search" />
                <input type="hidden" name="classgroup"  value="" autocapitalize="off" />
            </div>
            <a class="filter-search filter-default" href="#" title="${i18n().intro_filtersearch}">
                <span class="displace">${i18n().intro_filtersearch}</span>
            </a>
            <ul id="filter-search-nav">
                <li><a class="active" href="">${i18n().all_capitalized}</a></li>
                <@lh.allClassGroupNames vClassGroups! />
            </ul>
        </form>
    </fieldset>
</section> <!-- #search-home -->
```

This code lays out all of the formatting and markup, but the actual strings of text are retrieved from the ttl files, depending on the current language and locale.

8.2.7.4 Language-specific templates

Language-specific templates have been completely removed starting from the VIVO/Vitro 1.14.0 release.

All Freemaker templates are constructed like the one above; the text is merged with the markup at runtime.

8.2.7.5 Language in Java code

Java code has access to the same language properties that Freemarker accesses. Here is an example of using a language-specific string in Java code:

Excerpt from UserAccountsAddPageStrategy.java

```
FreemarkerEmailMessage email = FreemarkerEmailFactory.createNewMessage(vreq);
email.addRecipient(TO, page.getAddedAccount().getEmailAddress());
email.setSubject(i18n.text("account_created_subject", getSiteName()));
```

The ttl files contain these lines:

English language properties used in the example

```
prop-data:account_created_subject.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Your {0} account has been created."@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "account_created_subject" .
```

Note how the name of the VIVO site is passed as a parameter to the text message.

8.2.7.6 Language in JSPs

Up through VIVO release 1.14, no attempt has been made to add language support to JSPs.

8.2.7.7 Language in JavaScript files

To access string properties in JavaScript called from a template, assign the properties to variables in the Freemarker template, and then access those values from the JavaScript.

For example, the template can contain this:

Excerpt from page-home.ftl

```
<script>
    var i18nStrings = {
        countriesAndRegions: '${i18n().countries_and_regions}',
        statesString: '${i18n().map_states_string}',
</script>
```

And the script can contain this:

Excerpt from homePageMaps.js

```
    if ( area == "global" ) {
        text = " " + i18nStrings.countriesAndRegions;
    }
    else if ( area == "country" ) {
        text = " " + i18nStrings.statesString;
    }
```

8.2.8 Enabling Interface Languages in VIVO as an Administrator

If you are a VIVO site administrator and you simply want to enable one of the available languages[110] in your VIVO installation, the Quick Start guide should be all you need. If, however, you are interested in design details, then continue reading.

8.2.8.1 Quick Start

1. Stop the VIVO application
2. Update the **runtime.properties** file to:
    a. Enable language filtering

    ```
    RDFService.languageFilter = true
    ```

    b. Enabled desired languages
    ..such as the following if you wanted: English, Spanish, German, French and Portuguese

    ```
    languages.selectableLocales = en_US, es, de_DE, fr_CA, pt_BR
    ```

    Private-use subtags are also supported, meaning in the list of selectableLocales might be used local/institutional extension of the language such as fr_CA_x_uqam or de_DE_x_tib. However, those language files are not in the VIVO/Vitro codebase, meaning it is not possible just to enable those languages without copying of the directory with translations in those languages in the [VIVO_HOME] directory.

---

[110] https://github.com/vivo-project/Vitro-languages#available-language-files

8.2.8.2 VIVO i18n Design

One of the goals of the most recent design of VIVO's internationalization (i18n) implementation is to simplify the process for activating a multi-lingual VIVO site. Starting with the 1.12.0 release of VIVO, all supported languages are built into the application. On start-up, VIVO loads the UI labels ttl files, and annotation and other labels RDF for the languages enabled in the **runtime.properties** file; see Quick Start above.

When internationalization is enabled, additional logic is executed that filters content triples and selects fallback content in the case that there are no triples available for the preferred language/locale.

For example, if the site language is set to Canadian French ('fr_CA'), the search algorithm for returning triples for a specific label is as follows:

- Return label triples with language tag that matches the language ('fr') and locale ('CA'), if not found
- Return label triples with language tag that matches just the language ('fr'), if not found
- Return label triples with language tag that matches the language, but with a different locale (e.g. 'fr_FR'), if not found
- Return label triples with any or no language tag

8.2.9 Using VIVO's Internationalization (i18n) Features

Scope: This document explains how to use the internationalization (i18n) features in the VIVO front-end editing functionality.

For detailed explanations of how i18n functionality works in VIVO please consult the following :

- How VIVO supports languages (see page 133)

Generally the end user does not need to know how VIVO implements i18n functionality, however a basic familiarity with the underlying language feature is helpful in understanding what one can do to ensure correct usage.

8.2.9.1 Summary of language support

[copied here directly from Internationalization (see page 129)]

Internationalization

When a VIVO site supports a language other than English, that support includes:

- Text that is displayed in the VIVO pages. For example, menus, selections, prompts, tool-tips and plain text.
- Text from terms in the Ontology, which are frequently displayed as links or section headings. Text includes labels and annotations of properties and classes.

- Text values stored in the data. For example, if a book title is available in both French and English, a French-speaking user sees the French title. If a title is available only in English, it is displayed, without regard to the user's preference in languages.

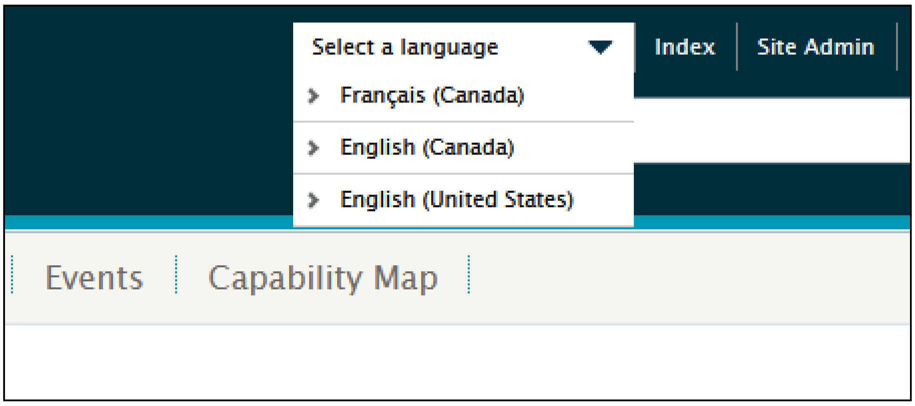
Languages can be selected in a variety of ways, depending on the installation parameters:

- A VIVO administrator can configure VIVO to use one of the supported languages.
- Different users may see different languages, depending on the settings in their web browser.
- Different users may select a language from a list of available languages.

VIVO language files are available for English (U.S. and Canadian), Spanish, Brazilian Portuguese, French (Canadian) and German. If you need support for another language, please inquire of the VIVO mailing lists, to see if another group has the files you need.

8.2.9.2 Selecting a language

If multiple languages are an option in your VIVO installation, you may toggle between the languages using the language selector in the top right of the screen, here we can see three language options for en_US, en_CA, and fr_CA :



8.2.9.3 Adding content in multiple languages

When adding new content to VIVO using the editing functionality, it is recommended that you enter as many versions of the text as you have available languages. So for example if you have a bilingual English-French interface, if you are creating a new entry for a person, any text should be entered both in the English and French interface.

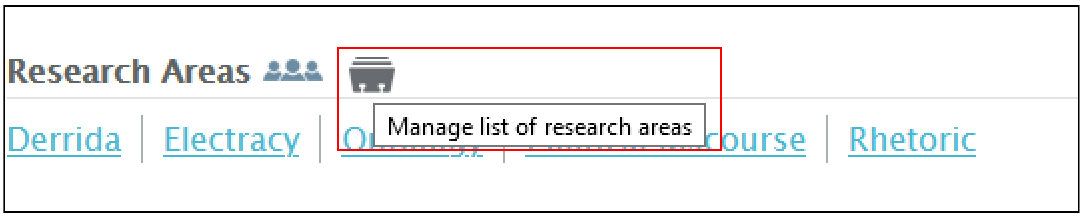
If text is added in only one language, i.e., English, the other language will look first for a French version, and if not available will present and display any available versions, following these rules:

For example, if the site language is set to Canadian French ('fr_CA'), the search algorithm for returning triples for a specific label is as follows:

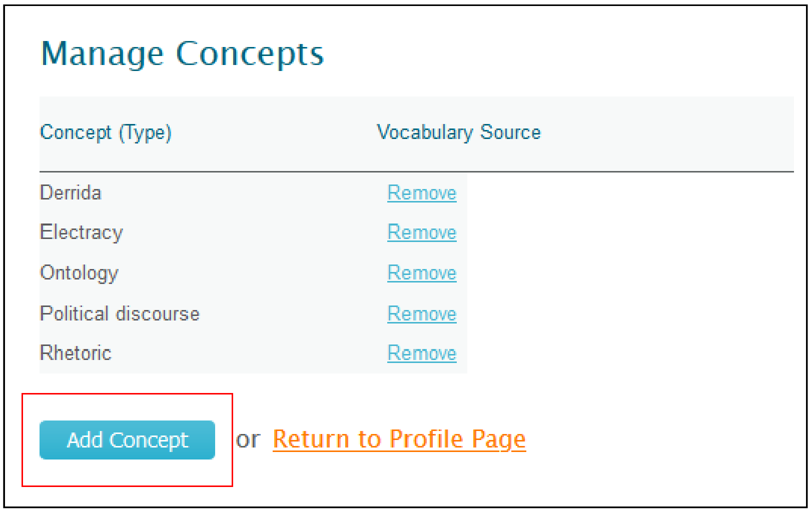
- Return label triples with language tag that matches the language ('fr') and locale ('CA'), if not found
- Return label triples with language tag that matches just the language ('fr'), if not found
- Return label triples with language tag that matches the language, but with a different locale (e.g. 'fr_FR'), if not found
- Return label triples with any or no language tag

Here is a step by step example of adding a new concept in both English and French.

1. Beginning in the English interface, from the profile of a professor, we will add a new concept "Phenomenology" to the list of research areas, by clicking on the 'manage list of research areas' icon on the profile page.


Research Areas
Manage list of research areas
Derrida
Electracy
Political discourse
Rhetoric

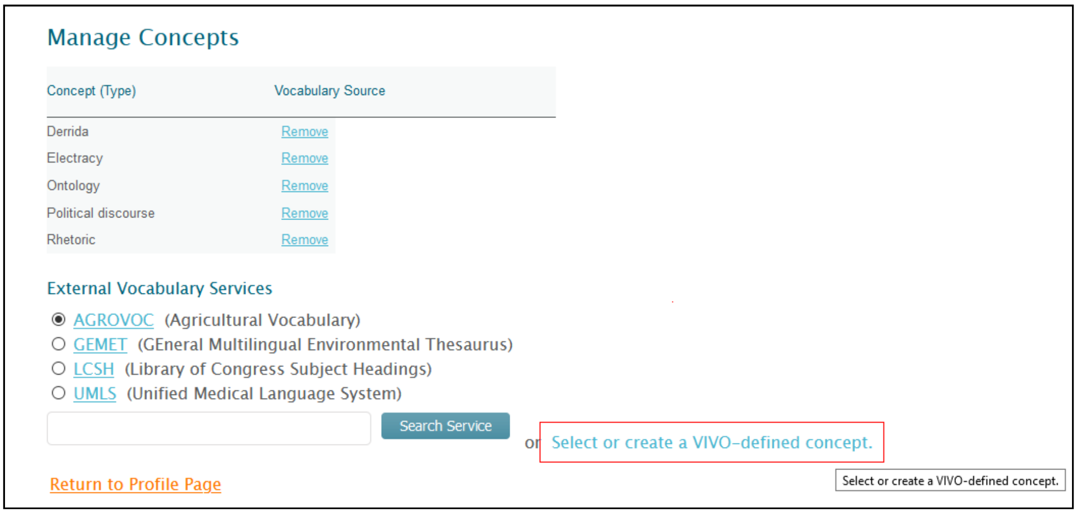
2. From the Manage Concepts screen click on the 'Add concept' button.


Manage Concepts

Concept (Type)	Vocabulary Source
Derrida	Remove
Electracy	Remove
Ontology	Remove
Political discourse	Remove
Rhetoric	Remove

Add Concept
or Return to Profile Page

3. Click on the link 'Select or create a VIVO-defined concept'.



Manage Concepts

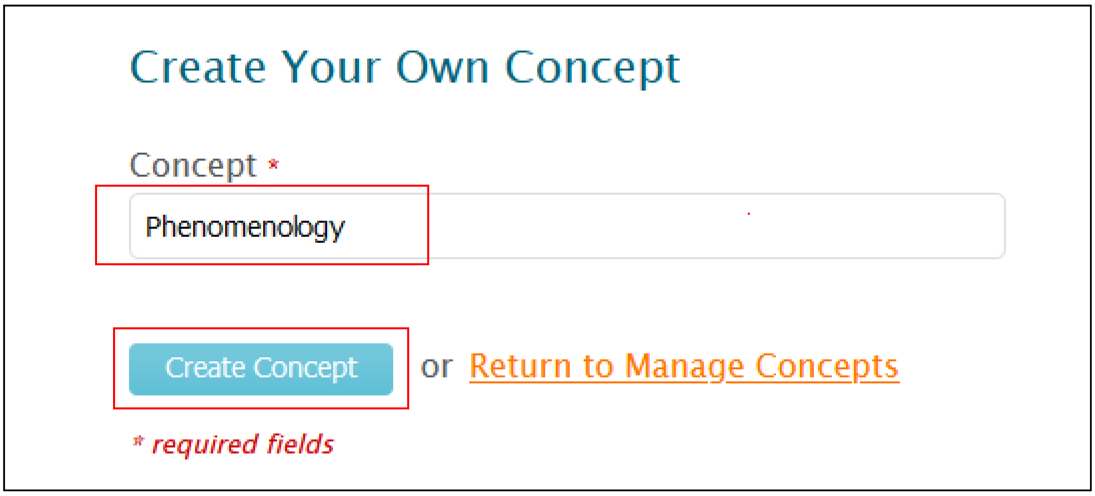
Concept (Type)	Vocabulary Source
Derrida	Remove
Electracy	Remove
Ontology	Remove
Political discourse	Remove
Rhetoric	Remove

External Vocabulary Services

- ● AGROVOC (Agricultural Vocabulary)
- ○ GEMET (GEneral Multilingual Environmental Thesaurus)
- ○ LCSH (Library of Congress Subject Headings)
- ○ UMLS (Unified Medical Language System)

Search Service
or Select or create a VIVO-defined concept.
Return to Profile Page
Select or create a VIVO-defined concept.

3. Add a new concept, here "Phenomenology", and click the 'Create Concept' button.

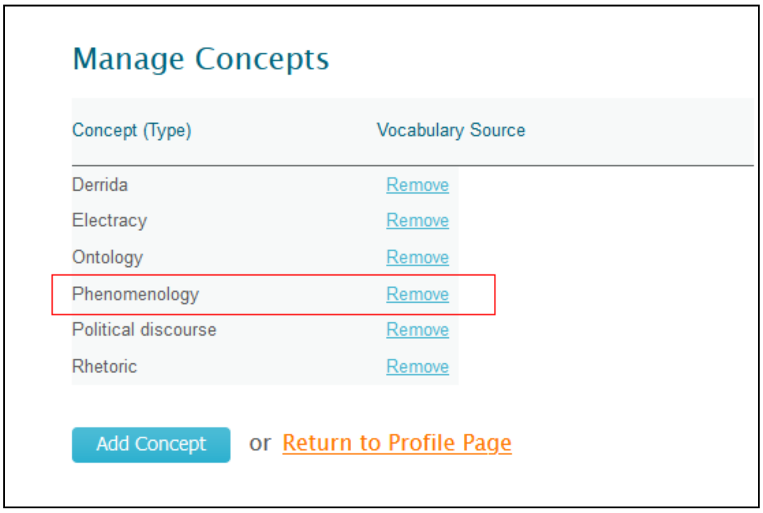


Create Your Own Concept

Concept * Phenomenology

Create Concept
or Return to Manage Concepts
* required fields

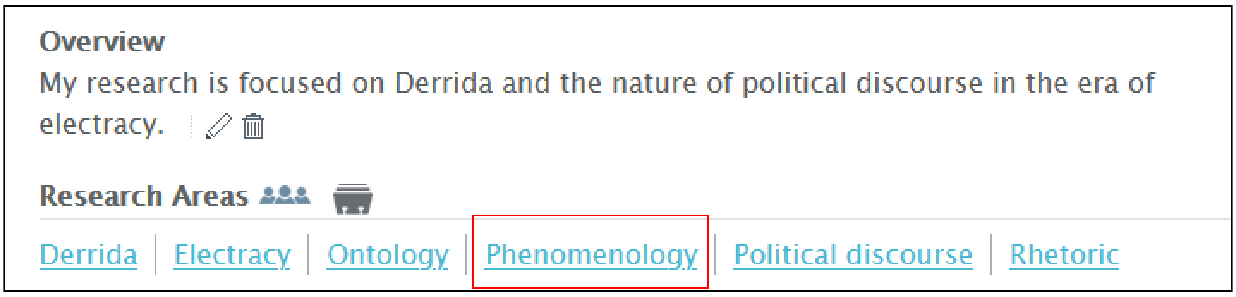
4. Returning to the Manage Concepts screen we can see that "Phenomenology" has been added to the concepts for this profile.



Manage Concepts

Concept (Type)	Vocabulary Source
Derrida	Remove
Electracy	Remove
Ontology	Remove
Phenomenology	Remove
Political discourse	Remove
Rhetoric	Remove

5. And returning to the profile page, we can see that the new concept now appears among the research areas for this professor.



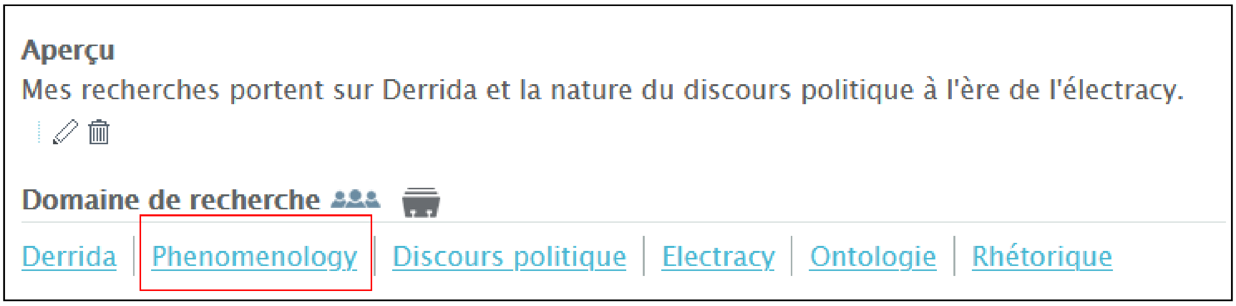
Overview

My research is focused on Derrida and the nature of political discourse in the era of electracy.

Research Areas

Derrida | Electracy | Ontology | Phenomenology | Political discourse | Rhetoric

6. If we toggle to the French language interface, we can see that the concept is listed, but in the English version in which it was created.



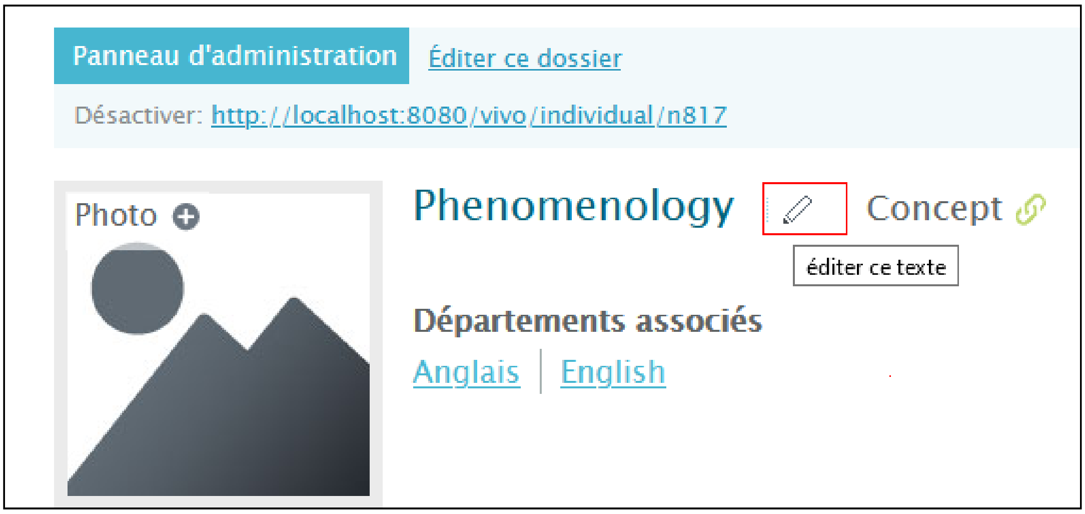
Aperçu

Mes recherches portent sur Derrida et la nature du discours politique à l'ère de l'électracy.

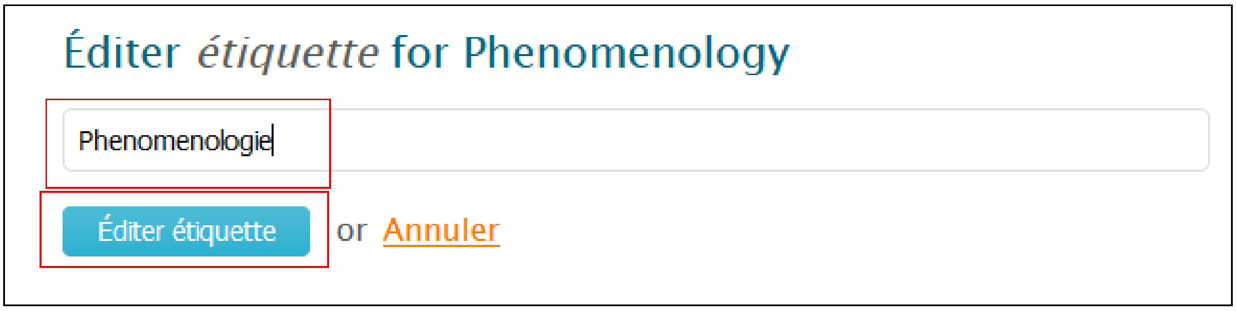
Domaine de recherche

Derrida | Phenomenology | Discours politique | Electracy | Ontologie | Rhétorique

7. To edit the concept so that it is displayed in French, click on the concept that needs to be translated to bring us to the screen for this concept. Make sure you are in the correct language context, here in French, and click on the icon to edit the concept text.


Panneau d'administration
Éditer ce dossier
Désactiver: http://localhost:8080/vivo/individual/n817
Photo
Phenomenology
Concept
éditer ce texte
Départements associés
Anglais
English

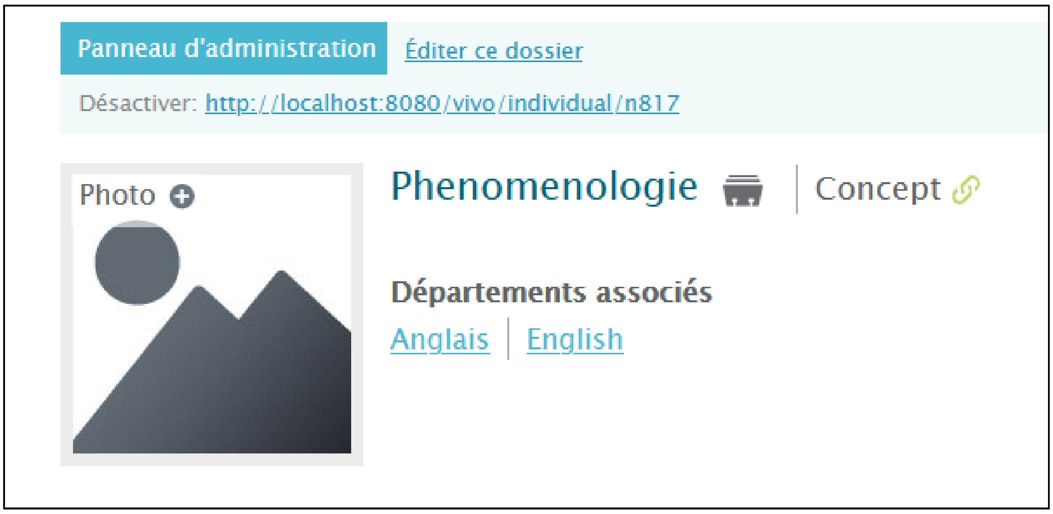
8. Add the equivalent French term "Phenomenologie", and save the change by clicking on the edit button (here 'Editer étiquette').


Éditer *étiquette* for Phenomenology

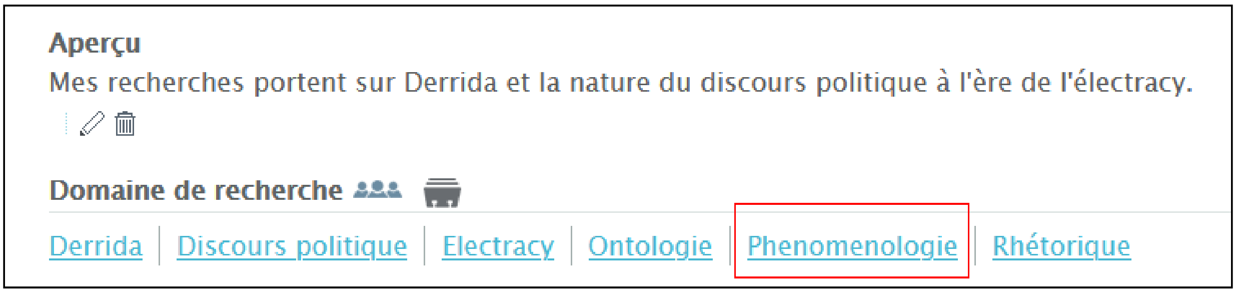
Phenomenologie

Éditer étiquette
or Annuler

9. Returning to the concept's screen, we can see that it is now displayed in French.


Panneau d'administration
Éditer ce dossier
Désactiver: http://localhost:8080/vivo/individual/n817
Photo
Phenomenologie
Concept
Départements associés
Anglais
English

10. And returning to the professor's profile we can now see the concept is displayed in French.



Aperçu

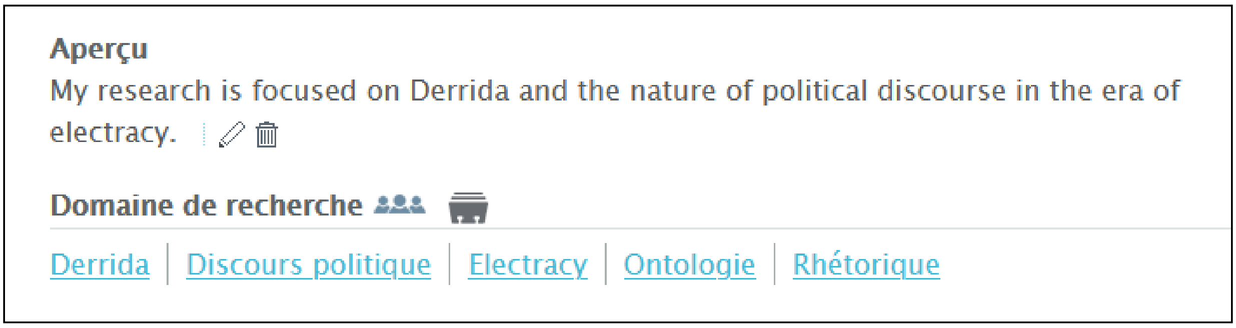
Mes recherches portent sur Derrida et la nature du discours politique à l'ère de l'électracy.

Domaine de recherche

Derrida | Discours politique | Electracy | Ontologie | Phenomenologie | Rhétorique

8.2.9.4 Editing content in multiple languages

When editing existing entries, you must also remember to edit the text in all available language contexts, use the language switcher to move from one language to the other.
Here is an example of a professor's profile where the Overview is displayed in English although we are in the French language display.



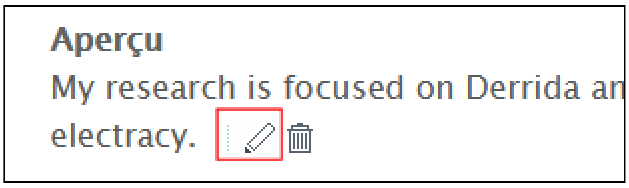
Aperçu

My research is focused on Derrida and the nature of political discourse in the era of electracy.

Domaine de recherche

Derrida | Discours politique | Electracy | Ontologie | Rhétorique

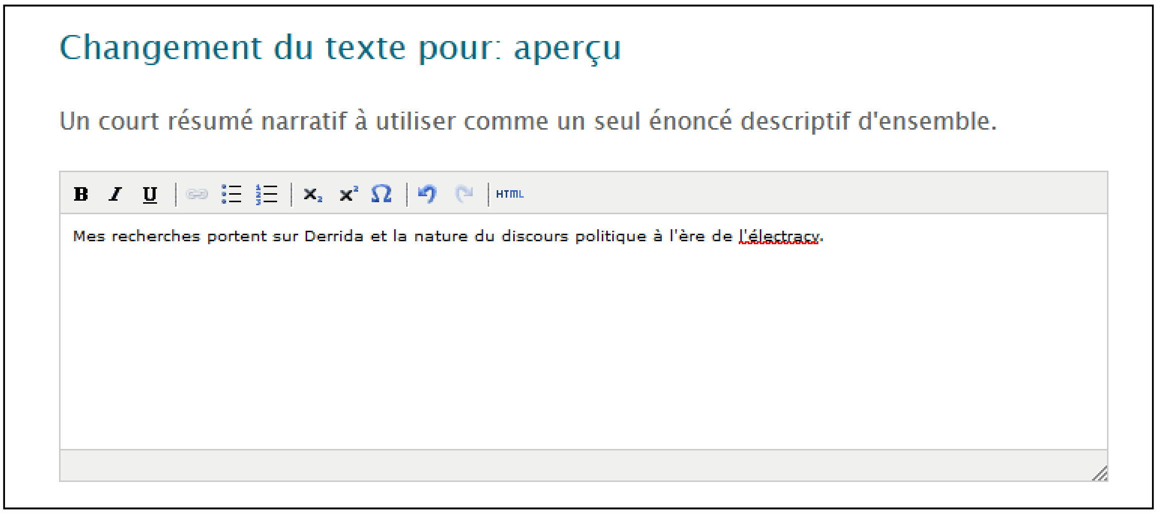
To change the description in 'Aperçu', click on the edit icon next to the description to bring you to the edit screen for the Overview/Aperçu and enter the French text



Aperçu

My research is focused on Derrida an

electracy.

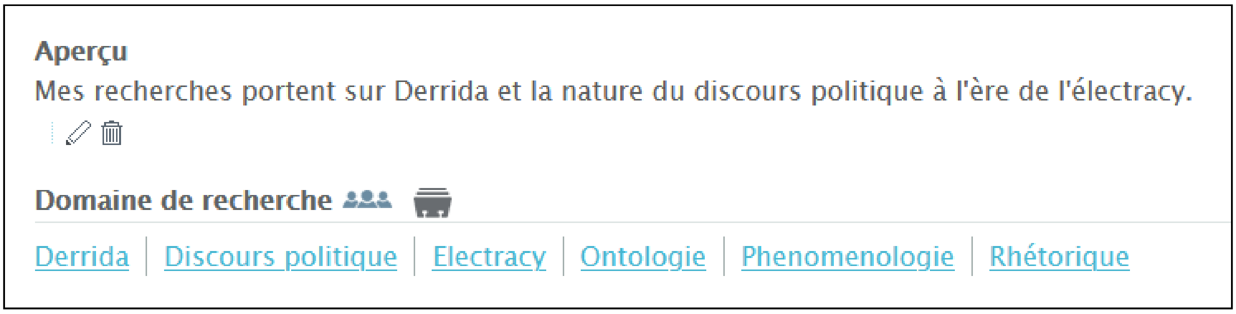


Changement du texte pour: aperçu

Un court résumé narratif à utiliser comme un seul énoncé descriptif d'ensemble.

Mes recherches portent sur Derrida et la nature du discours politique à l'ère de l'électracy.

After saving the changes, you can see the French language version displayed in 'Aperçu'.



Aperçu

Mes recherches portent sur Derrida et la nature du discours politique à l'ère de l'électracy.

Domaine de recherche

Derrida | Discours politique | Electracy | Ontologie | Phenomenologie | Rhétorique

8.2.9.5 Troubleshooting language related problems

In some situations, there may be unexpected behavior when displaying pages after adding or editing text in a multi-language installation. This may be for several reasons:

- faulty mapping in template or properties files
- cached pages in one language, lag in update of content
- necessity of rebuilding the index for content

In such cases, try refreshing your browser using Ctrl+F5. If this does not resolve the problem you should contact your VIVO administrator to rebuild the index or refresh the cache.

8.2.10 Vitro UI labels vocabulary

For the needs of supporting definition of user interface labels by using triplets preserved in a graph, the new VitroUILabels vocabulary has been defined. UI labels should be defined in accordance with this vocabulary.

The vocabulary defined PropertyKey class with internal structure containing five data properties: hasKey, hasApp, hasTheme, ftlUrl, hasPackage. Besides those five data properties, rdfs:label is used for defining UI label translations in different languages defined by the language tag assigned to rdfs:label. The values of properties hasKey, hasApp, and hasTheme, as well as language tag assigned to rdfs:label are used for selection of the UI label based on label key and the user environment (selected locale, theme, and application).

UILabelsVocabulary.ttl

```
@prefix owl:   <http://www.w3.org/2002/07/owl#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix prop-data: <http://vivoweb.org/ontology/core/properties/individual#> .
@prefix prop:  <http://vivoweb.org/ontology/core/properties/vocabulary#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix skos:  <http://www.w3.org/2004/02/skos/core#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

prop:hasPackage  rdf:type  owl:DatatypeProperty ;
    rdfs:domain  prop:PropertyKey ;
    rdfs:label   "has package" ;
    rdfs:range   xsd:string .

prop:hasKey  rdf:type  owl:DatatypeProperty ;
    rdfs:comment  "Value of the key" ;
    rdfs:domain   prop:PropertyKey ;
    rdfs:label    "Propertie file url " ;
    rdfs:range    xsd:string .

prop:hasTheme  rdf:type  owl:DatatypeProperty ;
    rdfs:domain  prop:PropertyKey ;
    rdfs:label   "has theme" ;
    rdfs:range   xsd:string .

prop:PropertyKey  rdf:type  owl:Class ;
    rdfs:label       skos:Concept ;
    rdfs:subClassOf  owl:Thing ;
    rdfs:subClassOf  skos:Concept .

prop:ftlUrl  rdf:type  owl:DatatypeProperty ;
    rdfs:comment  "Points to the FTL file containing the key" ;
    rdfs:domain   prop:PropertyKey ;
    rdfs:label    "ftl file url" ;
    rdfs:range    xsd:anyURI .

prop:hasApp  rdf:type  owl:DatatypeProperty ;
    rdfs:domain  prop:PropertyKey ;
    rdfs:label   "has application" ;
    rdfs:range   xsd:string .
```

8.3 Customizing the Interface

8.3.1 Introduction

8.3.1.1 Making changes to VIVO

The VIVO application is a popular tool for research networking. Most VIVO sites put their own changes into VIVO, in order to create a distinctive appearance, or to satisfy their particular needs.

VIVO supports an assortment of tools and techniques for making these changes. Some changes can be accomplished while VIVO is running, simply by setting values on a form. Other changes require you to add or modify configuration files that control the application. Still other changes are accomplished by editing the VIVO code, re-building, and re-deploying the application.

8.3.1.2 VIVO is already customized

Customization is built in to the heart of VIVO. VIVO itself is a customization of a more basic product called Vitro.

Here is how Vitro has been customized to become VIVO

Vitro	VIVO
No ontology	Includes an ontology for Research Networking
Minimal theme	Rich theme.
Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page
Default permissions	Display and editing permissions are customized, based on the ontology
Default editing forms	Editing is customized to the ontology
Default search index	Search index contains additional fields, specific to VIVO
Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.

Vitro	VIVO
In total: A general-purpose tool for working with Semantic Data.	**In total:** A specialized tool for Research Networking

8.3.2 Adding your own customizations

How do you add your changes to VIVO? Perhaps more important, how do you keep your changes when you upgrade to a newer release of VIVO?

8.3.2.1 Working in the GUI

When you use forms in VIVO, the values you enter are kept in the triple-store. They will be retained when you upgrade to a new release. If the new release uses a different format to store the values, your changes will be migrated to the new format.

8.3.2.2 RDF files

Some customizations require that you add or modify an RDF file in your VIVO home directory. In general, it's best to create a new file to contain the RDF statements, so you can easily carry your changes to a new VIVO release.

A "clean" build of VIVO will erase the RDF files in your VIVO home directory. You will need to re-create these files after the migration.

8.3.2.3 Changes to the source files

As with the RDF files, you should favor new files over changes to existing files. This will make it easier to carry your changes to a new release.

8.3.3 Tool summary

8.3.3.1 Required skills

The customization tools require different levels of knowledge. Some are as simple as filling out a web form. Most require the ability to write HTML, with additions from the Freemarker template engine. Some require Java programming.

As the tools are described, these terms will be used to specify the skills needed:

	Knowlege required
Basic	Requires an understanding of VIVO concepts.

Web development	The usual technologies for writing web sites, including HTML, CSS, and JavaScript. Knowledge of the Freemarker template engine.
RDF	Modify or create RDF data files, using RDF/XML, Turtle, or N3 format.
SPARQL	Create queries against the triple-store, using SPARQL.
Java	Create or modify Java code.
OpenSocial	Create or modify OpenSocial gadgets, written in JavaScript.

8.3.3.2 The tools

	What does it do?	**How?**	**Required skills**
Creating a custom theme (see page 210)	Create your own "brand" for VIVO. • Change colors, logo, headings, footers, and more.	CSS files, JavaScript files, and templates for HTML.	Web development
Annotations on the ontology (see page 166)	Control how data is displayed. • Property groups, labels, display order, hidden properties, and more.	Interactive.	Basic
Home page customizations (see page 154)	Choose from home page options. • Add a geographic focus map.	Edit your home page template to include a selection of sub-templates.	Web development
Menu and page management (see page 163)	Add new pages to VIVO. • Static pages, navigation pages, or dynamic reports.	Interactive.	Web development, optional SPARQL

Profiles for classes (see page 183)	Use one type of profile page for people and another for organizations.	Create page templates. Configure VIVO to associate them with classes.	Web development, RDF
Multiple profile types for foaf:Person (see page 240)	Provide a choice of formats for profile pages. • Each page owner selects the format for his own page.	Edit page templates. Perhaps connect to a Website image capture service.	Web development
Enriching profile pages with SPARQL queries (see page 235)	Display additional data on a profile page.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Enhancing page templates with SPARQL queries (see page 232)	Display additional data in any page template.	Write a SPARQL query. Modify a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom list views (see page 188)	Change how certain properties are displayed • Change the layout for that property • Display additional data with each value.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom short views (see page 196)	Change how search results are displayed • Display depends on the type of result (Person, Document, etc.). Also change display on index pages and browse pages.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom entry forms (see page 217)	Create data entry forms • Add or edit complex data structures.	Write a generator class in Java. Create a template for the editing form.	Web development, SPARQL, RDF, Java

Using Open Social Gadgets (see page 245)	Create optional content for profile pages. • Each page owner configures the gadgets for his own page.	Create gadgets from JavaScript, or install existing gadgets.	Web development, OpenSocial
Language support (see page 129)	Languages other than English • Use VIVO in Spanish • Allow viewers to choose their preferred language. • Implement other languages.	Create files of phrases in the desired language, or install existing files.	Basic

8.3.4 Home page customizations

8.3.4.1 Introduction

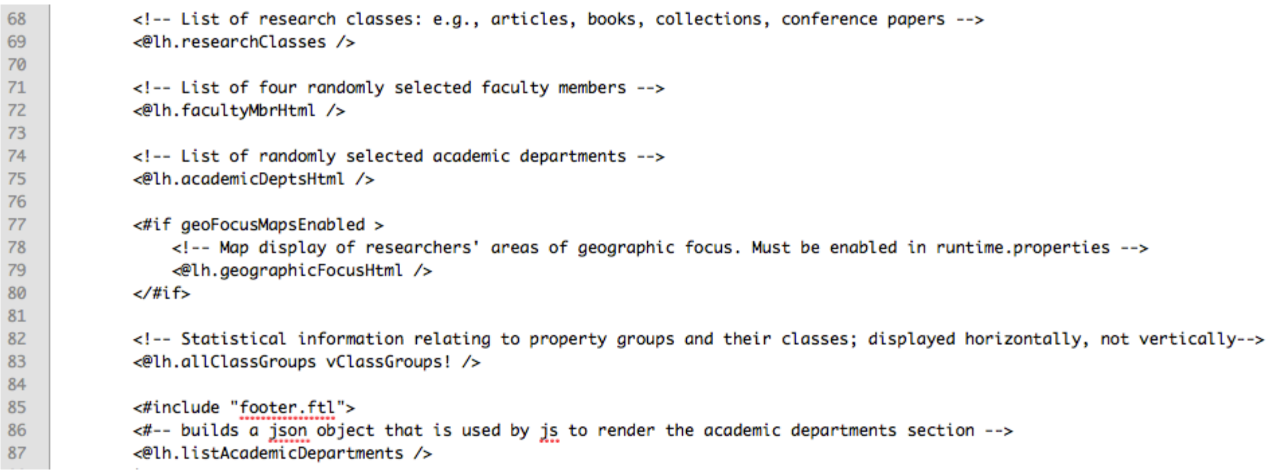
You can modify the "Research," "Faculty" and "Departments" sections of the home page, as well as expand the map section to include country-specific and state or province-specific maps.

8.3.4.2 The page-home.ftl Template File

The new sections of the home page are all referenced as macros in the page-home.ftl template file. The macros themselves are all located in the lib-home-page.ftl file, which is imported into the page-home.ftl file via this line:

```
<#import "lib-home-page.ftl" as lh>
```

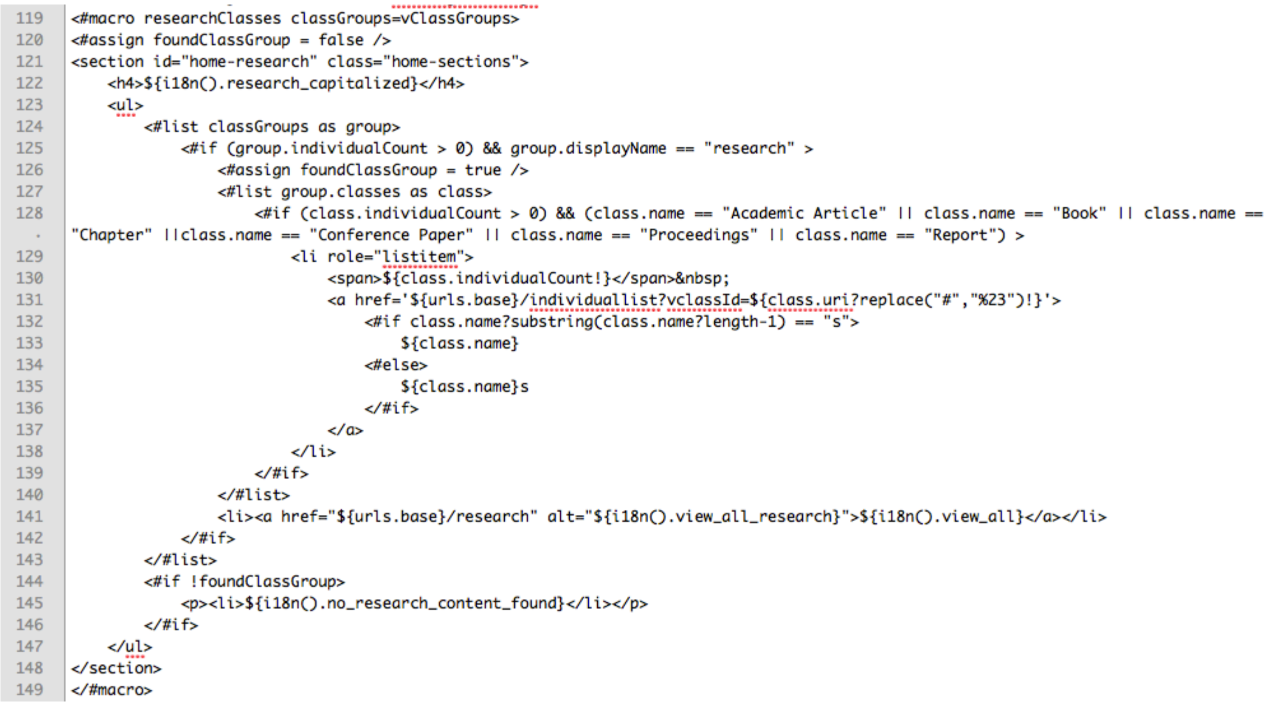
The code below is from the page-home.ftl template and shows how the macros are referenced. So, for example, if you wanted to modify the order in which these sections appear on the home page, you would move the macro references accordingly.



```
        <!-- List of research classes: e.g., articles, books, collections, conference papers -->
        <@lh.researchClasses />

        <!-- List of four randomly selected faculty members -->
        <@lh.facultyMbrHtml />

        <!-- List of randomly selected academic departments -->
        <@lh.academicDeptsHtml />

        <#if geoFocusMapsEnabled >
            <!-- Map display of researchers' areas of geographic focus. Must be enabled in runtime.properties -->
            <@lh.geographicFocusHtml />
        </#if>

        <!-- Statistical information relating to property groups and their classes; displayed horizontally, not vertically-->
        <@lh.allClassGroups vClassGroups! />

        <#include "footer.ftl">
        <#-- builds a json object that is used by js to render the academic departments section -->
        <@lh.listAcademicDepartments />
```

8.3.4.3 The Research Section

It's possible that your VIVO installation has defined some of its own classes within the Research Class group. Cornell's VIVO, for example, has a Library Collection class and a Media Contributions class. If your installation does include its own classes in this group, you can display these in the Research section of the home page by modifying the researchClasses macro in the lib-home-page.ftl file. As shown in line 128 below, the classes that get displayed are hard-coded into the macro. Simply exchange the name of your classes with some or all of the ones below. You could also add your classes to the existing list.

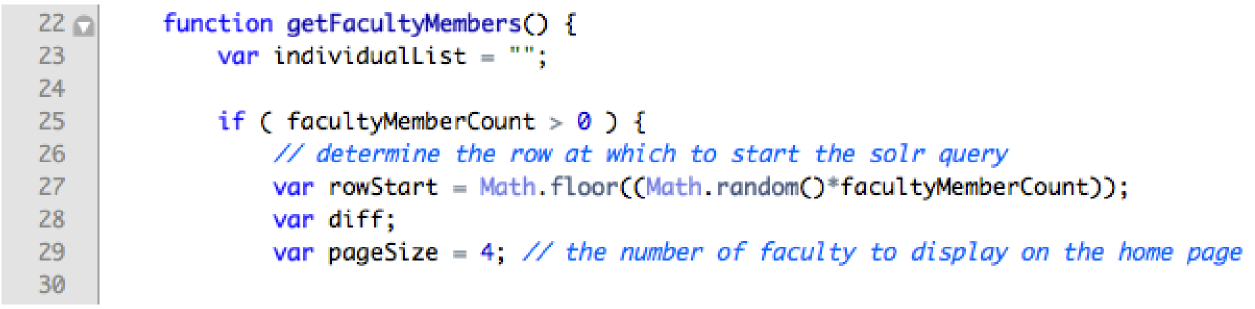


```
119  <#macro researchClasses classGroups=vClassGroups>
120  <#assign foundClassGroup = false />
121  <section id="home-research" class="home-sections">
122      <h4>${i18n().research_capitalized}</h4>
123      <ul>
124          <#list classGroups as group>
125              <#if (group.individualCount > 0) && group.displayName == "research" >
126                  <#assign foundClassGroup = true />
127                  <#list group.classes as class>
128                      <#if (class.individualCount > 0) && (class.name == "Academic Article" || class.name == "Book" || class.name ==
     "Chapter" ||class.name == "Conference Paper" || class.name == "Proceedings" || class.name == "Report") >
129                          <li role="listitem">
130                              <span>${class.individualCount!}</span>&nbsp;
131                              <a href='${urls.base}/individuallist?vclassId=${class.uri?replace("#","%23")!}'>
132                                  <#if class.name?substring(class.name?length-1) == "s">
133                                      ${class.name}
134                                  <#else>
135                                      ${class.name}s
136                                  </#if>
137                              </a>
138                          </li>
139                      </#if>
140                  </#list>
141                  <li><a href="${urls.base}/research" alt="${i18n().view_all_research}">${i18n().view_all}</a></li>
142              </#if>
143          </#list>
144          <#if !foundClassGroup>
145              <p><li>${i18n().no_research_content_found}</li></p>
146          </#if>
147      </ul>
148  </section>
149  </#macro>
```

It would be possible to display a random selection of classes rather than a hard-coded list, the same way that the Departments section displays a randomly selected list of academic departments. To do this, you would have to copy the macros and java script used for the academic departments, and then modify it accordingly so that it displays research classes. Refer to The Departments Section below for more details.

8.3.4.4 The Faculty Section

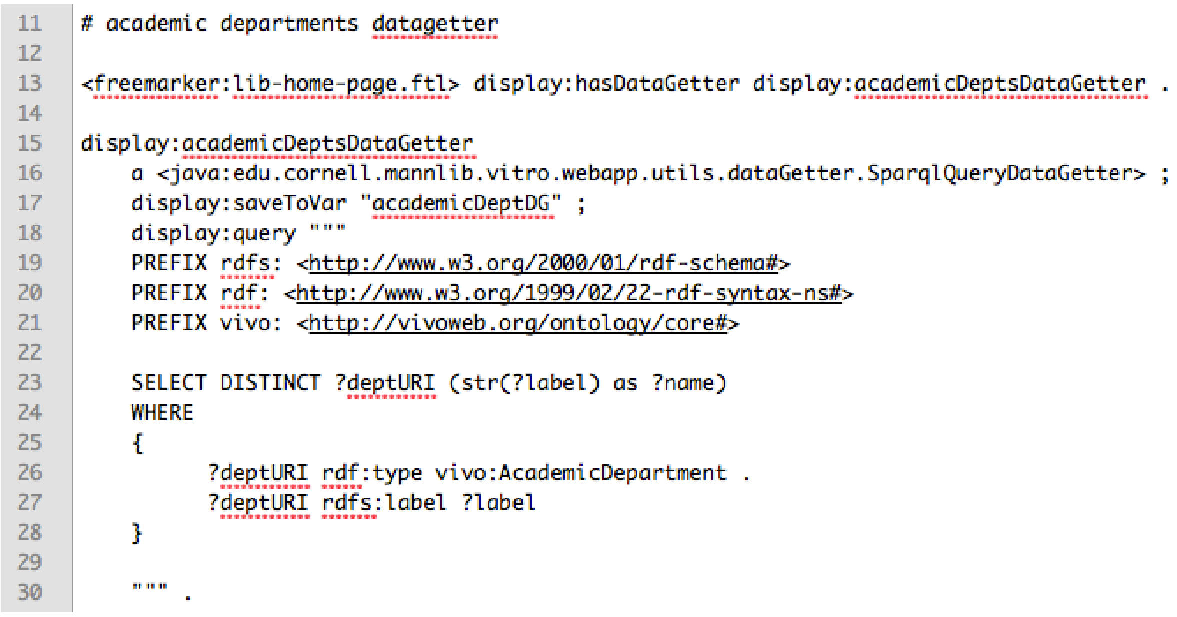
There's very little customization that can be done to the faculty section of the home page, excluding css changes and relocating the section to another part of the home page. The one configurable piece is the number of faculty members that get displayed. This change is made in the homePageUtils.js file. Locate the getFacultyMembers function and modify the pageSize variable (shown in line 29 below).



```
22  function getFacultyMembers() {
23      var individualList = "";
24
25      if ( facultyMemberCount > 0 ) {
26          // determine the row at which to start the solr query
27          var rowStart = Math.floor((Math.random()*facultyMemberCount));
28          var diff;
29          var pageSize = 4; // the number of faculty to display on the home page
30
```

8.3.4.5 The Departments Section

The list of academic departments is a randomly selected list that relies on a data getter as well as two macros in the lib-home-page.ftl file. The data getter is defined in the homePageDataGetters.n3 file. If you want to display something other than academic departments, you need to update the SPARQL query portion of the data getter, shown in lines 18-30 below. Substitute the class you want to display for vivo:AcademicDepartment.

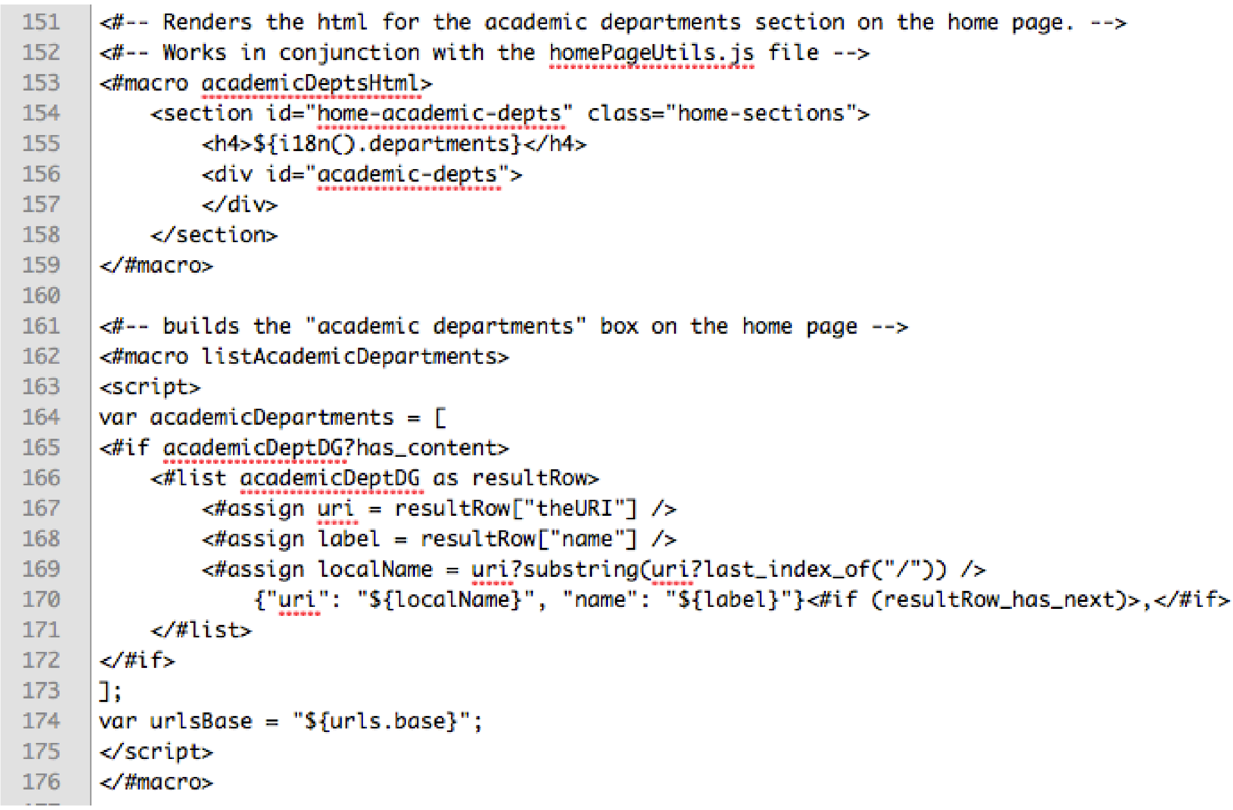


```
11  # academic departments datagetter
12
13  <freemarker:lib-home-page.ftl> display:hasDataGetter display:academicDeptsDataGetter .
14
15  display:academicDeptsDataGetter
16      a <java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter> ;
17      display:saveToVar "academicDeptDG" ;
18      display:query """
19      PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
20      PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
21      PREFIX vivo: <http://vivoweb.org/ontology/core#>
22
23      SELECT DISTINCT ?deptURI (str(?label) as ?name)
24      WHERE
25      {
26            ?deptURI rdf:type vivo:AcademicDepartment .
27            ?deptURI rdfs:label ?label
28      }
29
30      """ .
```

It is possible to expand the query to include more than one class. To do so without having to make any other macro or template changes, use UNION clauses in your query, as follows:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>

SELECT DISTINCT ?theURI (str(?label) as ?name)
WHERE
{{
   ?theURI rdf:type vivo:AcademicDepartment .
   ?theURI rdfs:label ?label .
}
UNION
{
   ?theURI rdf:type vivo:Association .
   ?theURI rdfs:label ?label .
}}
```

The following code snippet shows the two macros used to render the Departments section. If you change the data getter to use a different class, you do not have to change any variable or macro names. The only change you'll need to make is to the heading of this section so that it correctly reflects the class being displayed. Line 155 (below) is where you would make the change. (Note the use of the internationalization variable. As part of your change, you may want to update the i18n/all.properties file to include your new section heading.



```
151  <#-- Renders the html for the academic departments section on the home page. -->
152  <#-- Works in conjunction with the homePageUtils.js file -->
153  <#macro academicDeptsHtml>
154      <section id="home-academic-depts" class="home-sections">
155          <h4>${i18n().departments}</h4>
156          <div id="academic-depts">
157          </div>
158      </section>
159  </#macro>
160
161  <#-- builds the "academic departments" box on the home page -->
162  <#macro listAcademicDepartments>
163  <script>
164  var academicDepartments = [
165  <#if academicDeptDG?has_content>
166      <#list academicDeptDG as resultRow>
167          <#assign uri = resultRow["theURI"] />
168          <#assign label = resultRow["name"] />
169          <#assign localName = uri?substring(uri?last_index_of("/")) />
170              {"uri": "${localName}", "name": "${label}"}<#if (resultRow_has_next)>,</#if>
171      </#list>
172  </#if>
173  ];
174  var urlsBase = "${urls.base}";
175  </script>
176  </#macro>
```

8.3.4.6 The Geographic Focus Map

The new map on the home page uses circular markers to show the countries and regions that researchers in a VIVO installation have chosen as their areas of geographic focus (vivo:GeographicFocus). Clicking on a marker takes the user to that country or region's profile page, which shows the list of researchers in that location. The map is built using the Leaflet.js java script library, map tiles provided without charge by ESRI, and geographical data stored in a JSON file.

The map is enabled in the runtime.properties file. Include or uncomment the line:

```
homePage.geoFocusMaps=enabled
```

8.3.4.6.1 How the Map Works

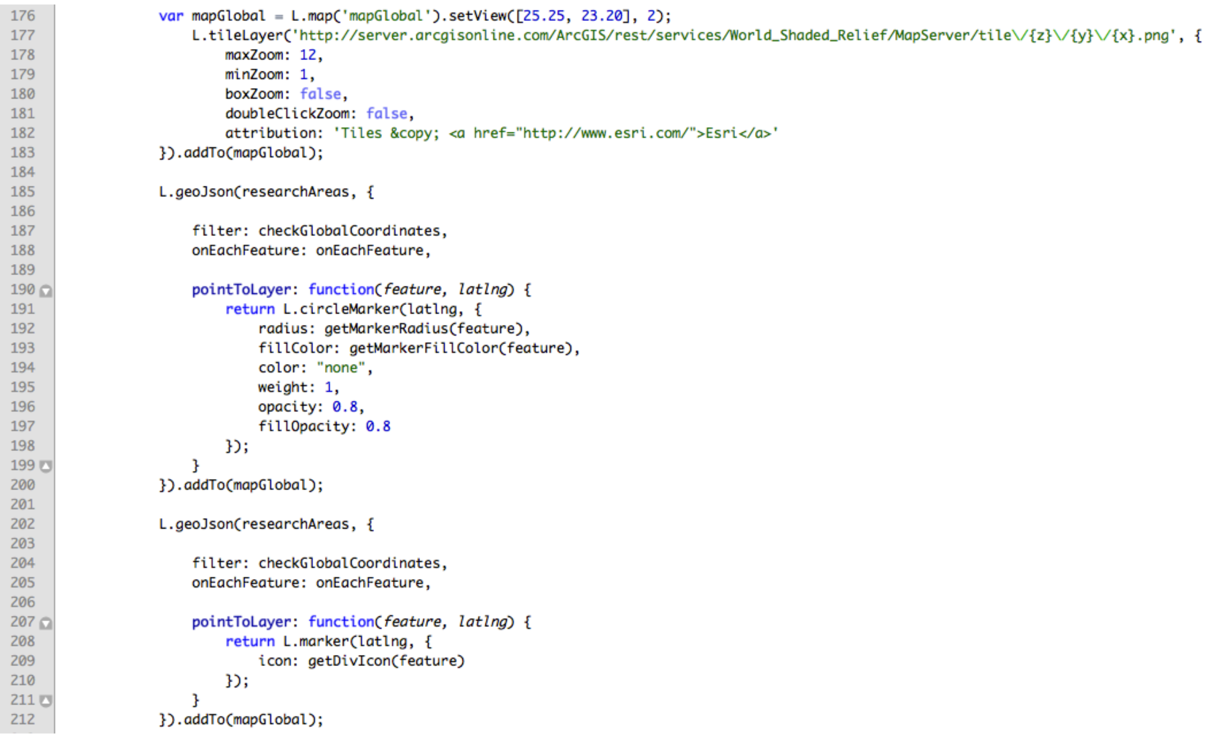
When the home page gets loaded, three java script files relating specifically to the map are sourced in: leaflet.js, latLongJson.js and homePageMaps.js. The first is the java script library that does the actual map rendering, from sourcing in the map tiles to placing the markers on the map. The second file contains a JSON array containing geographic data such as the names of countries and regions, their latitude and longitude, and some additional information that is used to build the GeoJSON object. The last file, homePageMaps.js, contains the functions that serve as the driver for rendering the map. The following outline covers the sequence of those events.

1) The getGeoJsonForMaps() function uses an AJAX request to call the GeoFocusMapLocations.java class. The purpose of this class is to run the SPARQL query that retrieves the names of the countries and regions that researchers have selected as areas of geographic focus as well the number of researchers associated with each area.

2) Once the SPARQL query results are returned to the getGeoJsonForMaps() function, it then parses the results and uses several function calls to build the GeoJSON array that gets used by the Leaflet java script. For example, the getLatLong() function call gets the longitude and latitude of a geographic area from the latLongJson.js file. The GeoJSON array, which is stored in a variable named "researchAreas," takes this format:

```
{ "type": "FeatureCollection",
  "features": [{'geometry': {'type': 'Point', 'coordinates': '-64.0,-34.0'},
                'type': 'Feature',
                'properties': {'mapType': 'global',
                               'popupContent': 'Argentina',
                               'html': '1',
                               'radius': '8,
                               'uri':
'http%3A%2F%2Faims.fao.org%2Faos%2Fgeopolitical.owl%23Argentina'}},
                {'geometry': {'type': 'Point', 'coordinates': '-2.0,54.0'},
                'type': 'Feature',
                'properties': {'mapType': 'global',
                               'popupContent': 'United Kingdom',
                               'html': '6',
                               'radius': '10,
                               'uri':
'http%3A%2F%2Faims.fao.org%2Faos%2Fgeopolitical.owl%23United_Kingdom'}},
                ... ]
}
```

3) Once the researchAreas variable is set, the buildGlobalMap() function is called. The main portion of that function is shown below:



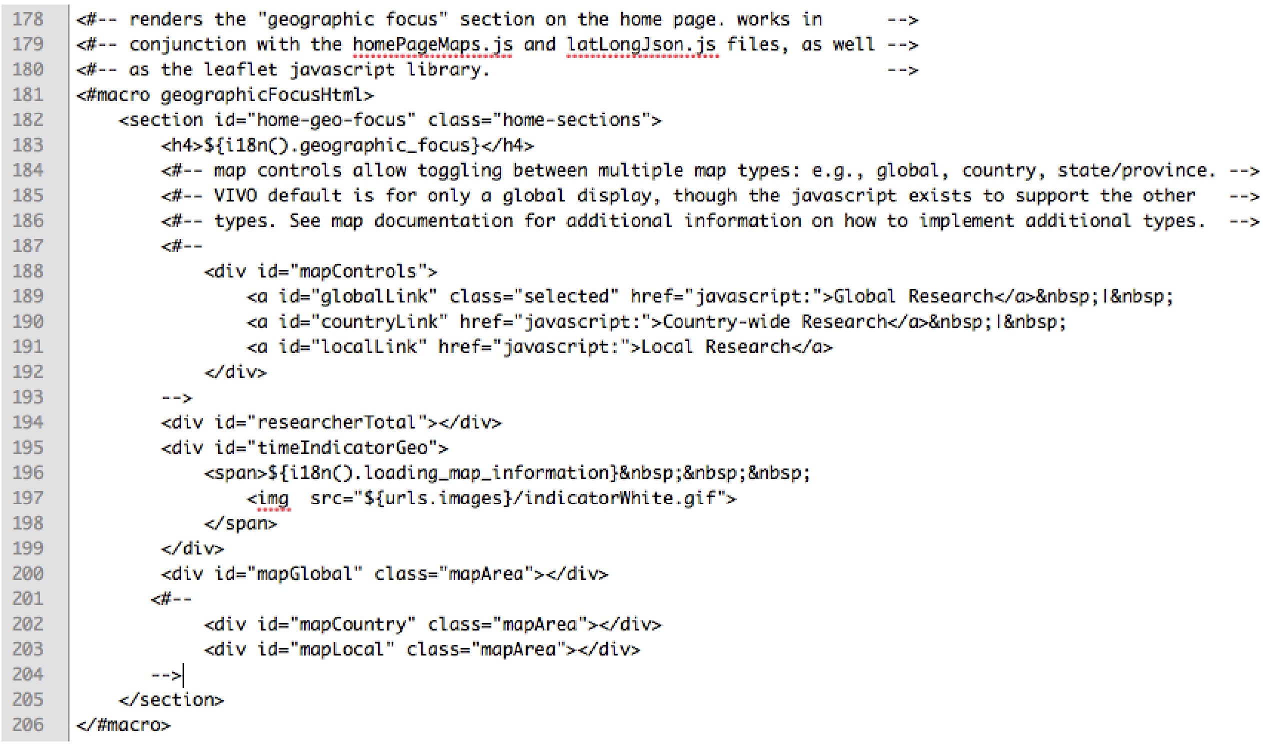
```
176         var mapGlobal = L.map('mapGlobal').setView([25.25, 23.20], 2);
177             L.tileLayer('http://server.arcgisonline.com/ArcGIS/rest/services/World_Shaded_Relief/MapServer/tile\/{z}\/{y}\/{x}.png', {
178                 maxZoom: 12,
179                 minZoom: 1,
180                 boxZoom: false,
181                 doubleClickZoom: false,
182                 attribution: 'Tiles &copy; <a href="http://www.esri.com/">Esri</a>'
183         }).addTo(mapGlobal);
184
185         L.geoJson(researchAreas, {
186
187             filter: checkGlobalCoordinates,
188             onEachFeature: onEachFeature,
189
190             pointToLayer: function(feature, latlng) {
191                 return L.circleMarker(latlng, {
192                     radius: getMarkerRadius(feature),
193                     fillColor: getMarkerFillColor(feature),
194                     color: "none",
195                     weight: 1,
196                     opacity: 0.8,
197                     fillOpacity: 0.8
198                 });
199             }
200         }).addTo(mapGlobal);
201
202         L.geoJson(researchAreas, {
203
204             filter: checkGlobalCoordinates,
205             onEachFeature: onEachFeature,
206
207             pointToLayer: function(feature, latlng) {
208                 return L.marker(latlng, {
209                     icon: getDivIcon(feature)
210                 });
211             }
212         }).addTo(mapGlobal);
```

Here are some key points to note about the previous code:

- The "L." references in the above code are calls to to Leaflet java script library.
- The setView function in line 176 uses latitude and longitude coordinates to center the display of the map.
- Also in line 176, 'mapGlobal' (in L.map('mapGlobal')...) is the name of the <div> element in which Leaflet will render the html for the map.

8.3.4.6.2 The geographicFocusHtml Macro

As noted earlier, the lib-home-page.ftl file contains the macros that are used to build the new sections on the home page. Here is the geographicFocusHtml macro:



```
178 <#-- renders the "geographic focus" section on the home page. works in      -->
179 <#-- conjunction with the homePageMaps.js and latLongJson.js files, as well -->
180 <#-- as the leaflet javascript library.                                     -->
181 <#macro geographicFocusHtml>
182     <section id="home-geo-focus" class="home-sections">
183         <h4>${i18n().geographic_focus}</h4>
184         <#-- map controls allow toggling between multiple map types: e.g., global, country, state/province. -->
185         <#-- VIVO default is for only a global display, though the javascript exists to support the other   -->
186         <#-- types. See map documentation for additional information on how to implement additional types.  -->
187         <#--
188             <div id="mapControls">
189                 <a id="globalLink" class="selected" href="javascript:">Global Research</a>&nbsp;|&nbsp;
190                 <a id="countryLink" href="javascript:">Country-wide Research</a>&nbsp;|&nbsp;
191                 <a id="localLink" href="javascript:">Local Research</a>
192             </div>
193         -->
194         <div id="researcherTotal"></div>
195         <div id="timeIndicatorGeo">
196             <span>${i18n().loading_map_information}&nbsp;&nbsp;&nbsp;
197                 <img  src="${urls.images}/indicatorWhite.gif">
198             </span>
199         </div>
200         <div id="mapGlobal" class="mapArea"></div>
201        <#--
202             <div id="mapCountry" class="mapArea"></div>
203             <div id="mapLocal" class="mapArea"></div>
204        -->
205     </section>
206 </#macro>
```

Note line 200: this is the <div> element where Leaflet renders the map.

8.3.4.6.3 Customizing the Look of the Map

There are three principal ways to customize the look of the map:

1. Change the source of the map tiles that provide the "atlas"
2. Change the colors of the markers
3. Change the size of the markers

8.3.4.6.3.1 Change the source of the map tiles

This is the most significant modification that you can make. The map currently uses tiles provided by ESRI, which has other map tiles for you to use. Mapquest is another source of free map tiles, as is Google. OpenCloud is a source of map tiles but they charge a small fee.

To change the tiles, you need to update the L.tileLayer definition in the buildGlobalMap() function. This is shown in line 177 above. Simply change the URL to the URL of the service providing your map tiles. (That service may also use a slightly different API.)

8.3.4.6.3.2 Change the colors of the markers

You can change the marker colors in the getMarkerFillColor() function in homePageMaps.js. Note that there are separate colors for countries and regions. If you do change the colors of the markers, you will also have to update the legend that appears in the lower left corner of the map. The circles in this legend are actually image files (map_legend_countries.png and map_legend_regions.png), so you will have to create new image files to match the colors you have chosen for markers.

8.3.4.6.3.3 Change the size of the markers

The size of the markers is the value that is set in the "radius" property in the GeoJSON array. This value is actually calculated in the GeoFocusMapLocations.java class. You can either update this class or add a new function to homePageMaps.js and modify the radius value in that java script file.

8.3.4.6.4 Enabling the Country and State/Province Maps

Currently, the home page map section only shows one map view: a global view with markers displayed for regions and countries. However, the code is available to include two additional views, one for a specific country and one for a specific state or province within a country. These are the steps you need to follow to implement the other two map views.

1. Update the geoFocusHtml macro
2. Update the coordinates in the setView() function
3. Update the getResearcherCount() function
4. Update the latLongJson.js file
5. Update the SPARQL query in the GeoFocusMapLocations.java class
6. Update your VIVO data as necessary

8.3.4.6.4.1 Update the geoFocusHtml macro

If you are using multiple map views, than you need to uncomment the mapControls <div> element in the geoFocusHtml macro (`<div id="mapControls">`). If you are only implementing two views (global and country), then you will want to ensure that the "localLink anchor tag is commented out (`<a id="localLink" href="javascript:">`). These anchor tags, along with corresponding java script in the homePageMaps.js file, allow the user to toggle between the implemented map views. (No change to the js file is necessary.)

Next you need to uncomment the <div> elements where the additional map views will be rendered: `<div id="mapCountry" class="mapArea">` and/or `<div id="mapLocal" class="mapArea">` . Again, only uncomment the <div> elements you are implementing.

8.3.4.6.4.2 Update the coordinates in the setView() function

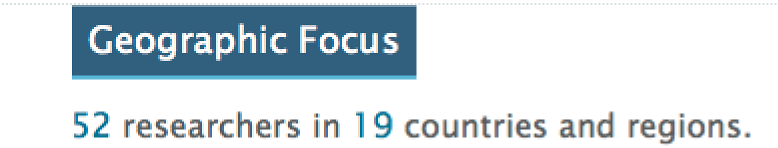
Besides the buildGlobalMap() function (discussed above), the homePageMaps.js file also includes buildCountryMap() and buildLocalMap() functions. These functions are very similar to the buildGlobalMap() function and work in the same way. When you are implementing a country map, you will want the map to be centered on that country.

```
var mapCountry = L.map('mapCountry').setView([46.0, -97.0], 3);
```

The coordinates above, `[46.0, -97.0]`, center the map on the United States. If you want this map to be centered on a different country, you will have to change these coordinates accordingly. The third value in the line of code above, `3`, is the zoom value and sets the default for when the map is loaded. Note that the default zoom value for the global map is `2` and the default for the local map could be any thing from `4` to `8` depending on the location you are displaying.

8.3.4.6.4.3 Update the getResearcherCount() function

For all three types of views, the map includes summary text that shows the total number of researchers and geographical areas in the results, as show below:



Geographic Focus

52 researchers in 19 countries and regions.

Depending on the map views you implement, and the actual country or areas they display, you may want to modify the wording that gets displayed here. This is done in the getResearcherCount() function in of the homePageMaps.js file. (Note that the text here uses internationalization variables, so you may need to update the i18n/all.properties file as well.)

8.3.4.6.4.4 Update the latLongJson.js file

The latLongJson.js file contains data for countries, transnational regions and states within the United States. Therefore, if your installation wants to implement a country map other than the U.S., you will need to update the latLongJson.js file to include the necessary data. For example, if the country to be displayed is Australia, the latLongJson.js file would need to include data on the states and territories of Australia. The JSON array in this file takes data in this format:

```
{"name": "Victoria", "data": {"mapType": "country", "geoClass": "state", "latitude":
"-37.4713", "longitude": "144.7851"}}
```

Note that the mapType corresponds to the map view, in this case "country" as opposed to "global, while the geoClass corresponds (loosely) to the VIVO ontology class. ("Loosely," because the class is actually "StateOrProvince.")

Implementing a state/province map would mean updating the latLongJson.js file to include data for the geographic areas within a state. For U.S. states, examples would include counties, townships and even more general areas such as the Hudson or Mohawk valleys in New York (two areas of geographic focus for Cornell researchers). In this third case the mapType must be set to "local."

8.3.4.6.4.5 Update the SPARQL query in the GeoFocusMapLocations.java class

For performance and practical reasons, the SPARQL query in the GeoFocusLocations.java class excludes states and provinces. (Since only the global map is displayed by default, there is no reason to include state and provinces in the query results.) To update the query to include states and provinces, simply remove this line from the query:

```
FILTER (NOT EXISTS {?location a core:StateOrProvince})
```

If you want to implement a state/province map, you may need to update the query further to ensure that the local geographic areas are included in the query results. Although there are VIVO classes for counties and "populated places," your VIVO installation might have additional refinements to the ontology.

8.3.4.6.4.6 Update your VIVO data as necessary

The SPARQL query in the GeoFocusLocations.java class does not simply return a count for the numbers of researchers that have selected a country (for example) as an area of geographic focus. The query also roll-ups the counts for "child" locations into the "parent" location. For example, if 5 researchers have the U.S. as their area of geographic focus, and another 5 researchers have individual states as their focus, the query will return a count of 10 for the U.S. This is accomplished through the vivo:geographicallyContains object property. Similarly, country counts are rolled up into regional counts through the geo:hasMember object property. *It's possible that you will need to curate your VIVO data to ensure that the necessary object property relationships exist in your installation. This is especially true with the local geographical areas.*

8.3.5 Menu and page management

8.3.5.1 Overview

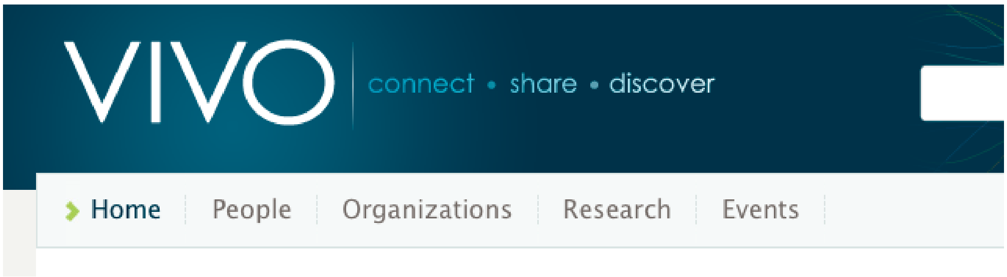
8.3.5.1.1 What can it do for you?

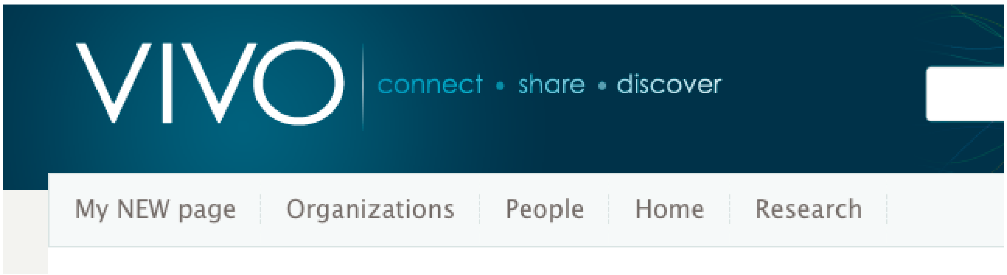
- —Create “browse” pages, static pages, or pages that display the results of a query (reports)
- —Remove existing pages
- —Manipulate the page menu

It's easy to surmise that Page Management only allows you to make changes to the menu. And it's true that you can create new menu pages, rearrange the menu, or remove items from it.

But you can also use Page Management to create pages that aren't in the menu. You assign a simple URL to each page, so you can link to them from your other pages. The content of the pages can be simple HTML, the results of a SPARQL query, or a "browse" page for individuals in VIVO.

8.3.5.1.2 Before and After


VIVO
connect • share • discover
Home
People
Organizations
Research
Events


VIVO
connect • share • discover
My NEW page
Organizations
People
Home
Research

8.3.5.1.3 What do you need to know?

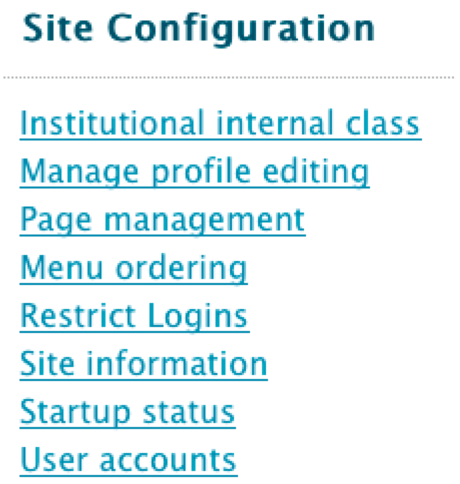
- —How to follow the GUI for page management
- —Optional – how to write Freemarker templates
- —Optional – how to write SPARQL queries

8.3.5.1.4 Getting started

VIVO comes with a set of managed pages, including the ones that you see in the menu on each page.

8.3.5.2 What to do

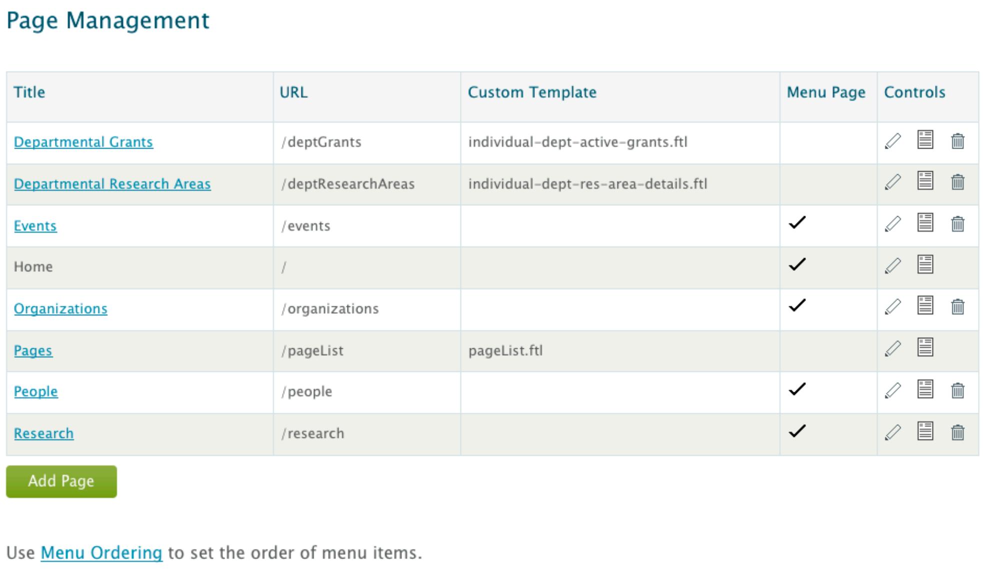
Go to the ***Site Admin*** page, and choose ***Page management***.



Site Configuration

- Institutional internal class
- Manage profile editing
- Page management
- Menu ordering
- Restrict Logins
- Site information
- Startup status
- User accounts

Use the links provided to create new pages, or edit existing ones.

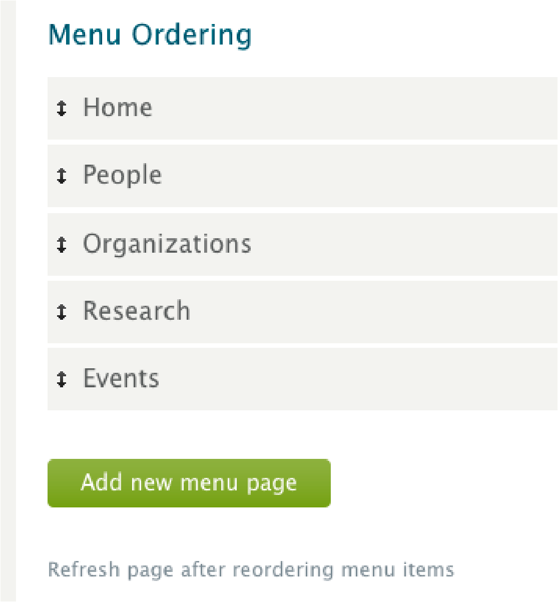


Page Management

Title	URL	Custom Template	Menu Page	Controls
Departmental Grants	/deptGrants	individual-dept-active-grants.ftl		✎ ▤ 🗑
Departmental Research Areas	/deptResearchAreas	individual-dept-res-area-details.ftl		✎ ▤ 🗑
Events	/events		✓	✎ ▤ 🗑
Home	/		✓	✎ ▤
Organizations	/organizations		✓	✎ ▤ 🗑
Pages	/pageList	pageList.ftl		✎ ▤
People	/people		✓	✎ ▤ 🗑
Research	/research		✓	✎ ▤ 🗑

Use Menu Ordering to set the order of menu items.

Click on the Menu Ordering link to re-arrange the menu. Drag entries up or down to establish the order you want. When you refresh the page, or go to another, you will see your changes in the menu.


Menu Ordering
↕ Home
↕ People
↕ Organizations
↕ Research
↕ Events
Add new menu page

Refresh page after reordering menu items

8.3.6 Annotations on the ontology

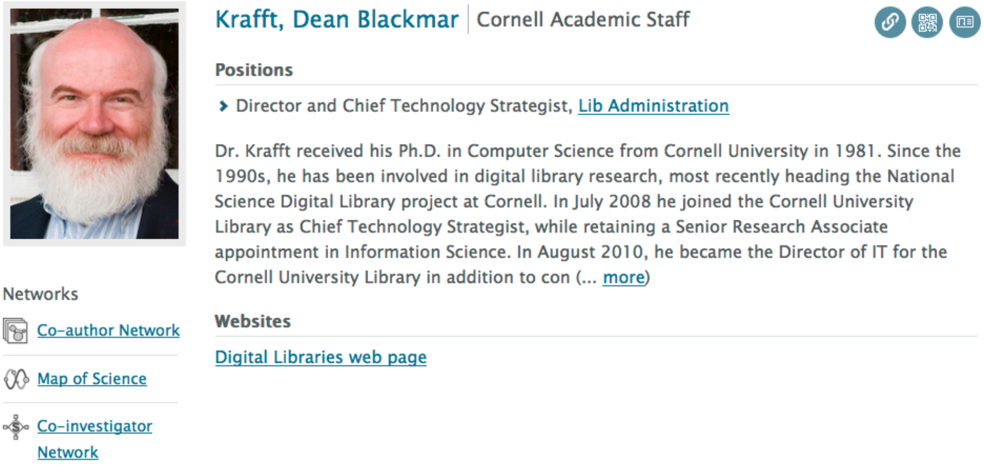
- Edit property groups (see page 166)
- Edit the appearance of properties (see page 169)
- Create and edit faux properties (see page 172)
- Edit class groups (see page 177)
- Edit the appearance of classes (see page 180)

8.3.6.1 Edit property groups

8.3.6.1.1 Overview

8.3.6.1.1.1 Before and After

Rename "Affiliation" to "Allegiances", and change the order of "Publications" and "Research".



Krafft, Dean Blackmar | Cornell Academic Staff

Positions

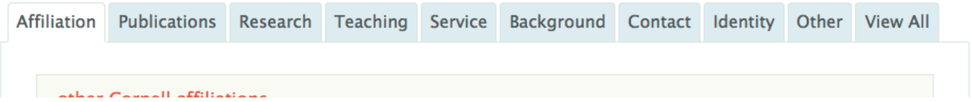
Director and Chief Technology Strategist, Lib Administration

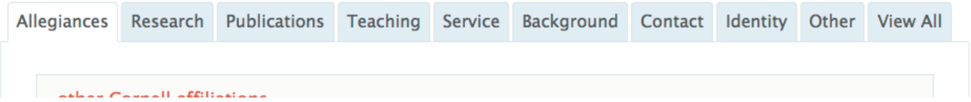
Dr. Krafft received his Ph.D. in Computer Science from Cornell University in 1981. Since the 1990s, he has been involved in digital library research, most recently heading the National Science Digital Library project at Cornell. In July 2008 he joined the Cornell University Library as Chief Technology Strategist, while retaining a Senior Research Associate appointment in Information Science. In August 2010, he became the Director of IT for the Cornell University Library in addition to con (... more)

Websites

Digital Libraries web page

Networks
Co-author Network
Map of Science
Co-investigator Network


Affiliation
Publications
Research
Teaching
Service
Background
Contact
Identity
Other
View All


Allegiances
Research
Publications
Teaching
Service
Background
Contact
Identity
Other
View All

8.3.6.1.1.2 What do you need to know?

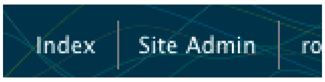
How to follow the GUI for property groups.

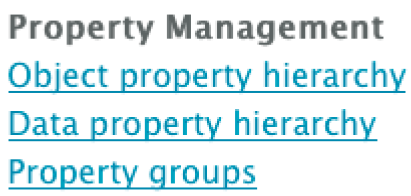
8.3.6.1.1.3 Getting started

VIVO comes with a default set of property groups. You can rename them, reorder them, create new groups or delete existing ones. You can also move properties from one group to another, but that is covered in Edit the appearance of properties (see page 169).

8.3.6.1.2 What to do

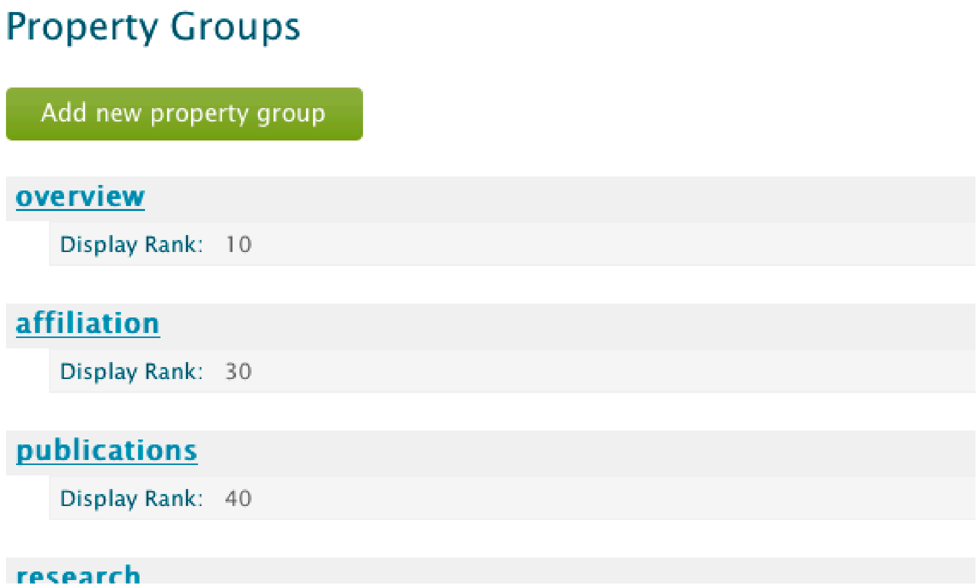
From the VIVO ***Site Admin*** page, navigate to the ***Property Groups*** page.


Index
Site Admin
ro



Property Management
Object property hierarchy
Data property hierarchy
Property groups

You can add a new property group, or click on the name of an existing group to edit it.



Property Groups

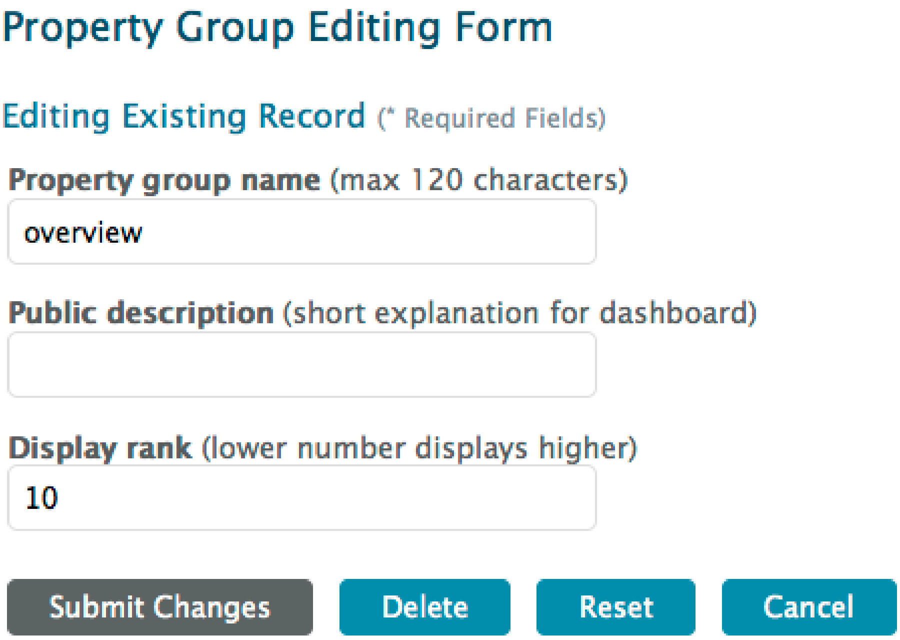
overview
Display Rank: 10

affiliation
Display Rank: 30

publications
Display Rank: 40

research

You can change the name of a group, change its display rank, or even delete it.



Property Group Editing Form

Editing Existing Record (* Required Fields)

Property group name (max 120 characters)

overview

Public description (short explanation for dashboard)

Display rank (lower number displays higher)

10

When an profile page is displayed, the property groups are shown in order of ascending display rank.

Properties that aren't included in any of the groups are displayed on the profile page as part of the group named `Other`. This is not an actual property group; it is simply a display convention.

If you delete a property group, the properties that were in it will be displayed in `Other`, until you assign them to new groups.

8.3.6.2 Edit the appearance of properties

8.3.6.2.1 Overview

8.3.6.2.1.1 What can it do for you?

Change how VIVO displays the properties of an individual.

For each property, you can change

- the display label
- the public and private descriptions
- which property group it belongs to
- the display rank within the property group
- who can see the values
- who can edit the values
- whether the values will be published in linked open data requests
- whether the display will be collated by sub-properties (object properties only)

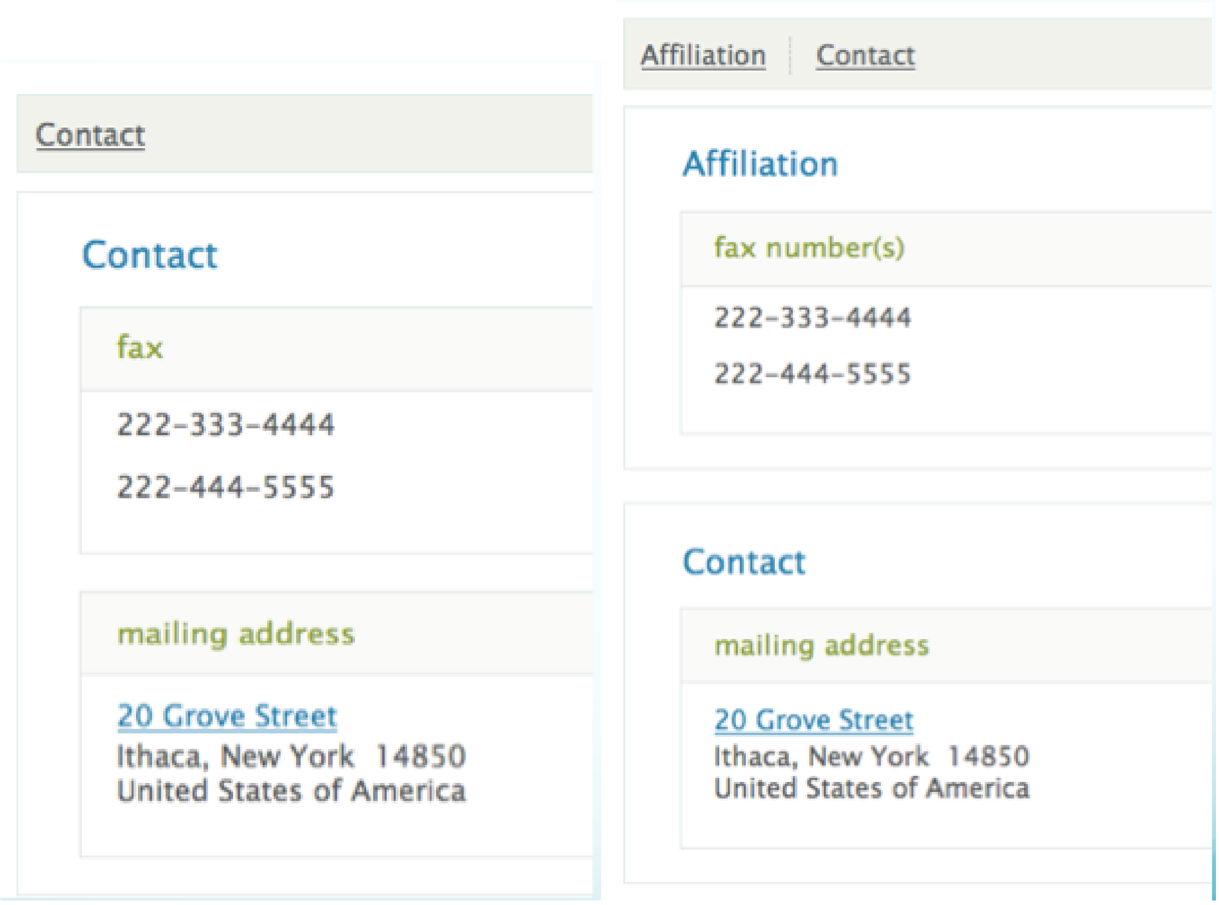
You can also change things like the namespace and parent property, but these are actually changes to the ontology.

Notice that there are necessary differences between the editing options for a data property and those for an object property. Since an object property describes the relationship between two individuals, it is richer than a data property which has only a text value.

The property editing form also allows you to assign a custom entry form to a property, as described in Customize: date entry forms (see page 166)

8.3.6.2.1.2 Before and After

Modify the "fax" property, changing the display label to "fax number(s)", and moving it to the "Affiliation" property group.


Contact
Contact
fax
222-333-4444
222-444-5555
mailing address
20 Grove Street
Ithaca, New York 14850
United States of America
Affiliation
Contact
Affiliation
fax number(s)
222-333-4444
222-444-5555
Contact
mailing address
20 Grove Street
Ithaca, New York 14850
United States of America

8.3.6.2.1.3 What do you need to know?

How to follow the GUI for property editing.—

8.3.6.2.1.4 Getting started

VIVO comes with a default set of properties. You can edit them to suit your display requirements, or make more extensive modifications, by customizing the ontology.

8.3.6.2.2 What to do

There are several ways to navigate to the ***Property Editing Form*** for a particular property. Perhaps the most common way is to show the profile page for an individual, turn on ***verbose property display***,

click on the name of the property you want to edit,

from the ***Property Control Panel***, choose to edit the property

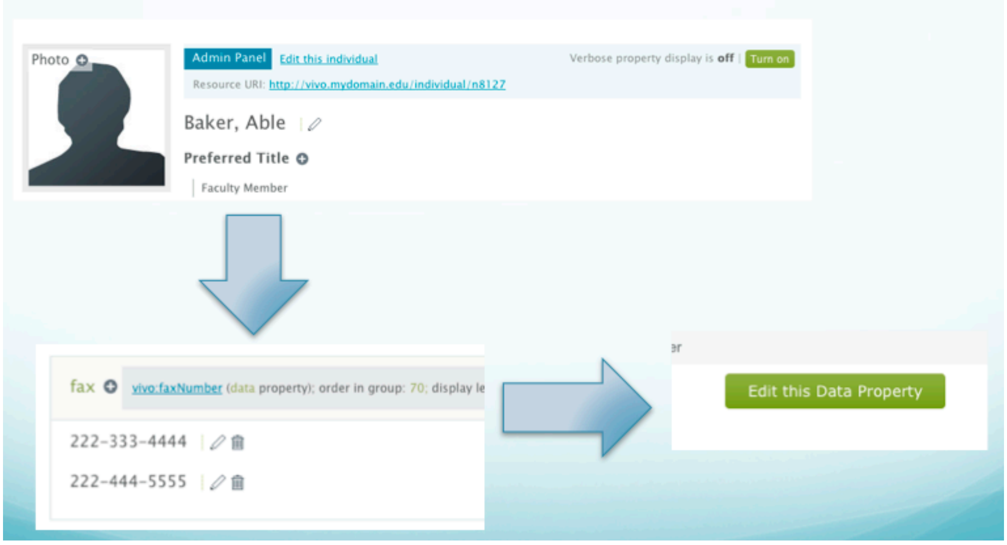
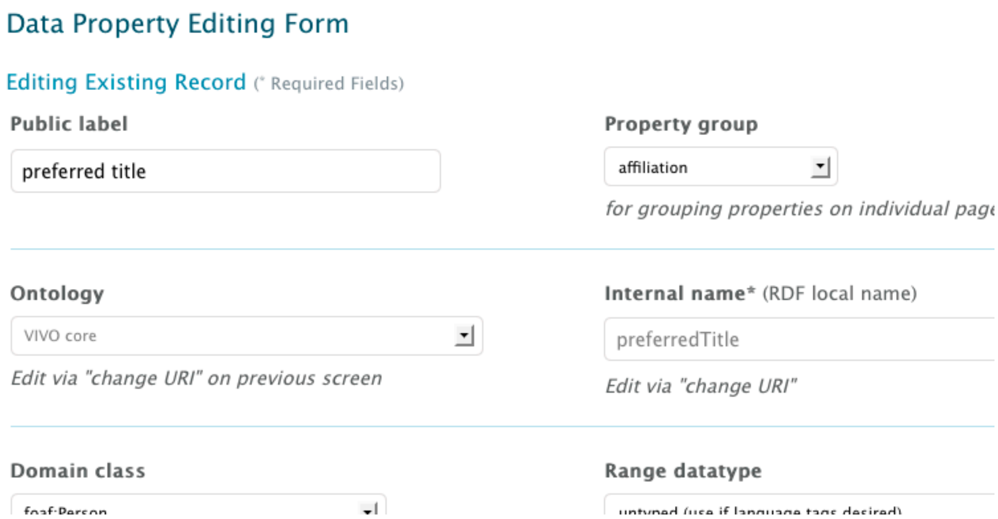


Photo
Admin Panel
Edit this individual
Verbose property display is off
Turn on
Resource URI: http://vivo.mydomain.edu/individual/n8127
Baker, Able
Preferred Title
Faculty Member
fax
vivo:faxNumber (data property); order in group: 70; display le
222-333-4444
222-444-5555
Edit this Data Property

When the property editing form appears, make the changes you want to see, and click on `Submit Changes`. Navigate to a profile page and you will see the effect of your changes immediately.



Data Property Editing Form

Editing Existing Record (* Required Fields)

Public label
preferred title

Property group
affiliation
for grouping properties on individual page

Ontology
VIVO core
Edit via "change URI" on previous screen

Internal name* (RDF local name)
preferredTitle
Edit via "change URI"

Domain class
foaf:Person

Range datatype
untyped (use if language tags desired)

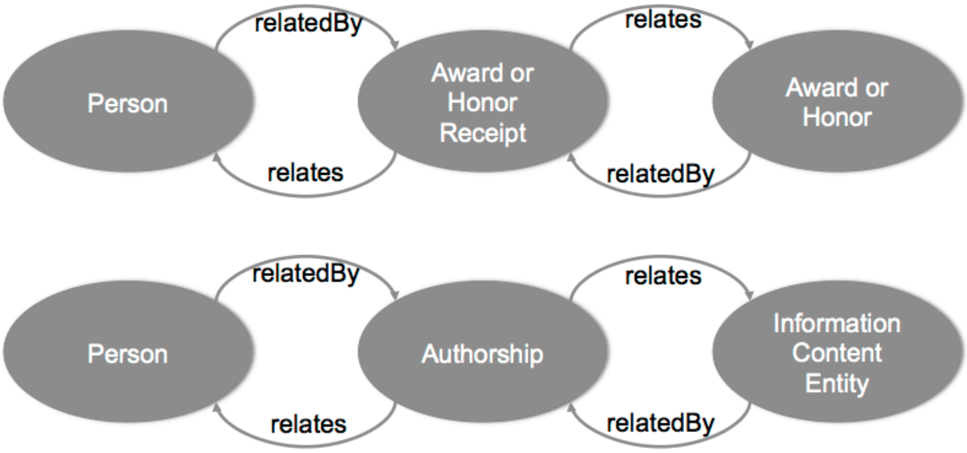
8.3.6.2.3 Notes

Properties may appear differently depending to someone who is authorized to edit them. Try logging out to see how your changes will appear to the general user.

8.3.6.3 Create and edit faux properties

8.3.6.3.1 Overview

The emphasis in ontology design has been fewer properties connecting more classes. For example, we see that the relationship between a Person, an Authorship, and an Article is very similar to the relationship between a Person, an Award Receipt and an Award. Similarly, a data property can be re-used by a set of classes and its subclasses.


relatedBy
relates
Person
Award or Honor Receipt
relates
relatedBy
Award or Honor
relatedBy
relates
Person
Authorship
relates
relatedBy
Information Content Entity

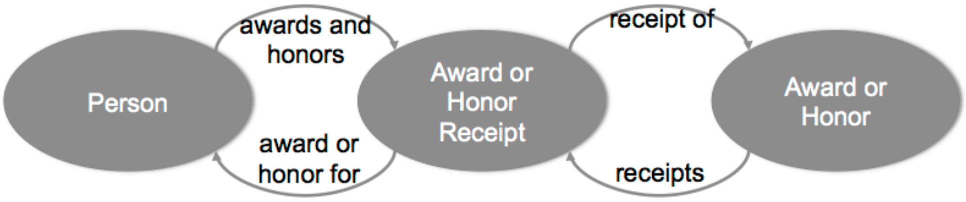
The profile pages in VIVO have been organized by properties. This re-use of properties makes it difficult to organize information on the page. Not only do we want to see at a glance the difference between an authorship and a received award; we may also want to display them in different areas of the page, using different custom views, etc.

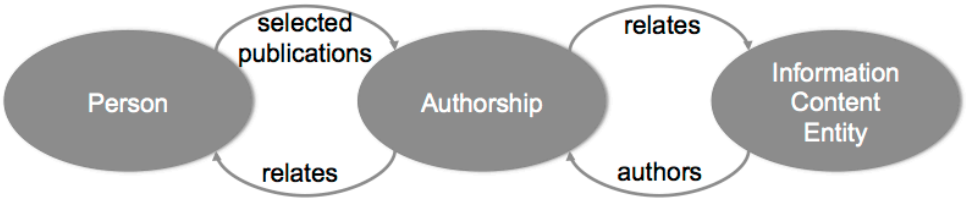
VIVO allows us to create "faux" properties, as restrictions on object or data properties. The faux property has the same property URI as its base property. Range of faux data properties is always the same as range of base data property. Domain and range of faux object properties and domain of faux data properties are limited to not disjoint classes. Once we have established these criteria, we can assign display properties to the faux property, just as if it were its own property with its own URI.

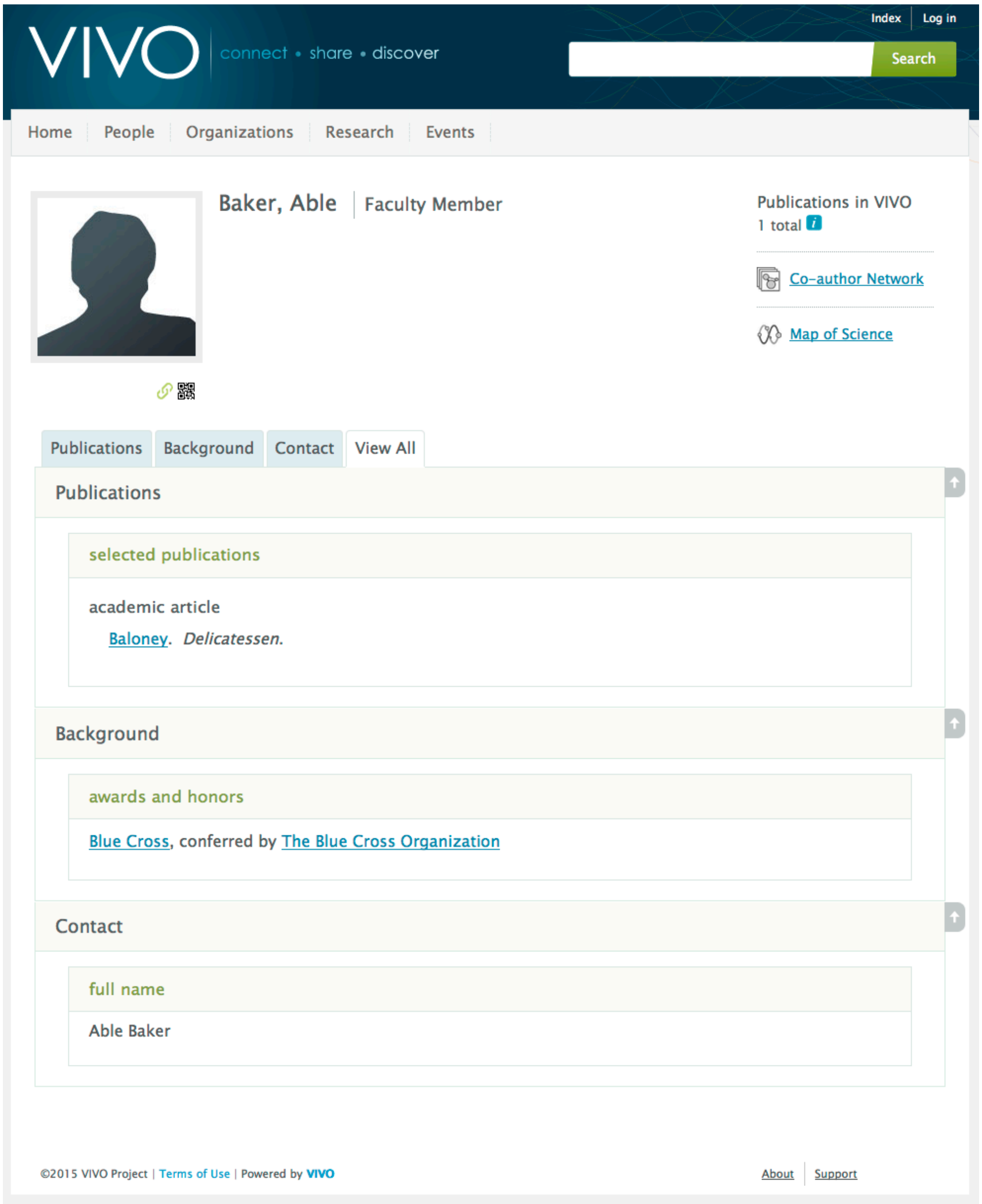
In this way, we can define faux properties as follows:

URI	Domain	Range	label	property group
relatedBy	Person	Award or Honor Receipt	awards and honors	Background
relatedBy	Person	Authorship	selected publications	Publications
relatedBy	Award or Honor	Award or Honor Receipt	receipts	Overview
relatedBy	Information Content Entity	Authorship	authors	Overview
relates	Award or Honor Receipt	Person	award or honor for	Overview
relates	Award or Honor	Award or Honor Receipt	receipt of	Overview

Now, these same relationships display quite differently:


Person
awards and honors
award or honor for
Award or Honor Receipt
receipt of
receipts
Award or Honor


Person
selected publications
relates
Authorship
relates
authors
Information Content Entity


VIVO
connect • share • discover
Index
Log in
Search
Home
People
Organizations
Research
Events

Baker, Able | Faculty Member

Publications in VIVO
1 total

Co-author Network

Map of Science

Publications
Background
Contact
View All

Publications

selected publications

academic article

Baloney. *Delicatessen.*

Background

awards and honors

Blue Cross, conferred by The Blue Cross Organization

Contact

full name

Able Baker

©2015 VIVO Project | Terms of Use | Powered by VIVO

About | Support

In general, it makes sense to partition all of a property into faux properties. Then the base property is set to be invisible (except to the root user), and the faux properties display the desired information.

8.3.6.3.2 What do you need to know?

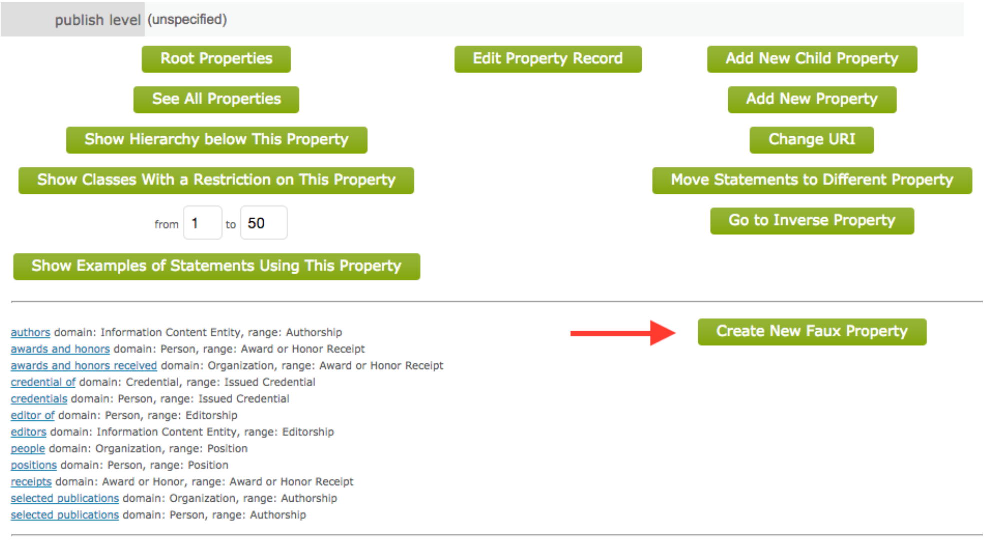
How to follow the GUI for faux properties.

8.3.6.3.3 Getting started

VIVO comes with a default set of faux properties. You can edit them to suit your display requirements, or make more extensive modifications.

8.3.6.3.4 Creating a faux property

First, navigate to the control panel for the property (object or data) you want to create a faux property for. The control panel can be accessed by clicking **Object Property Hierarchy** or **Data Property Hierarchy** on the Site Administration Page, then clicking on an object property, i.e. related by, or in the case Data Property Hierarchy by clicking on a data property, i.e. additional name. The Create New Faux Property button can be found next to a list of that property's existing faux properties.



publish level (unspecified)

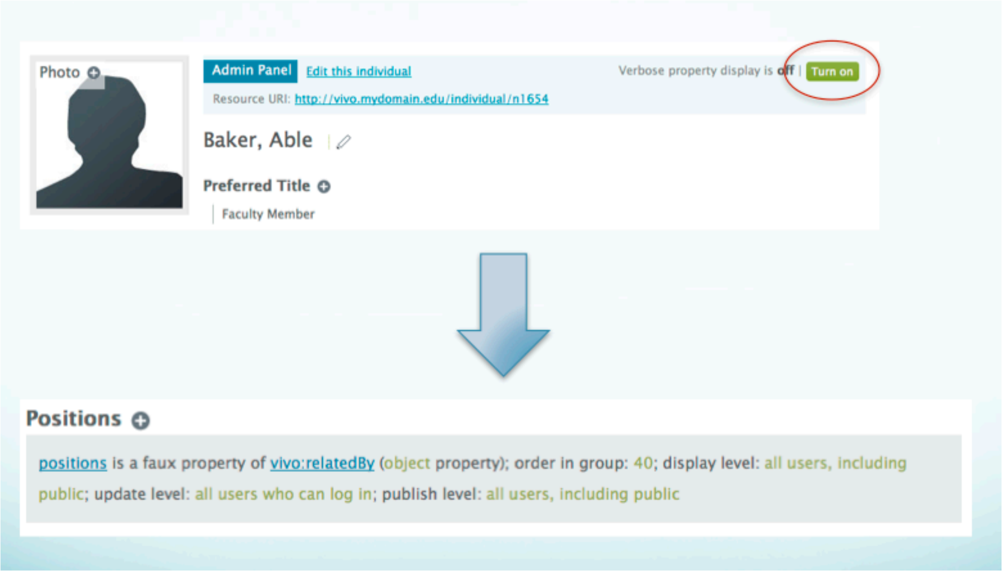
from 1 to 50

authors domain: Information Content Entity, range: Authorship
awards and honors domain: Person, range: Award or Honor Receipt
awards and honors received domain: Organization, range: Award or Honor Receipt
credential of domain: Credential, range: Issued Credential
credentials domain: Person, range: Issued Credential
editor of domain: Person, range: Editorship
editors domain: Information Content Entity, range: Editorship
people domain: Organization, range: Position
positions domain: Person, range: Position
receipts domain: Award or Honor, range: Award or Honor Receipt
selected publications domain: Organization, range: Authorship
selected publications domain: Person, range: Authorship

Root Properties
See All Properties
Show Hierarchy below This Property
Show Classes With a Restriction on This Property
Show Examples of Statements Using This Property
Edit Property Record
Add New Child Property
Add New Property
Change URI
Move Statements to Different Property
Go to Inverse Property
Create New Faux Property

8.3.6.3.5 Editing a faux property

There are several ways to navigate to the Faux Property Editing Form. Perhaps the most common way is to show the profile page for an individual, and turn on **verbose property display**.



Resource URI: http://vivo.mydomain.edu/individual/n1654

Baker, Able

Preferred Title

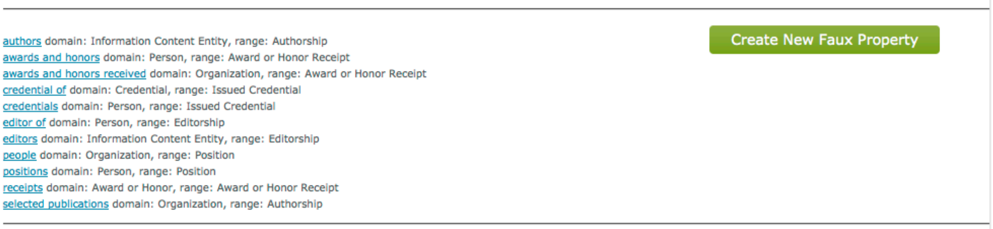
Faculty Member

Positions

positions is a faux property of vivo:relatedBy (object property); order in group: 40; display level: all users, including public; update level: all users who can log in; publish level: all users, including public

Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on

In this example, we see that `positions` is a faux property of `vivo:relatedBy`. If you click on the link for `positions`, you will see the Faux Property Editing Form for `positions`. On the other hand, if you click the link for `vivo:relatedBy`, you will see the Object Property Editing Form for `vivo:relatedBy`. In addition to the parameters for `vivo:relatedBy`, you will also see links to each of the faux properties that are based on it:



authors domain: Information Content Entity, range: Authorship
awards and honors domain: Person, range: Award or Honor Receipt
awards and honors received domain: Organization, range: Award or Honor Receipt
credential of domain: Credential, range: Issued Credential
credentials domain: Person, range: Issued Credential
editor of domain: Person, range: Editorship
editors domain: Information Content Entity, range: Editorship
people domain: Organization, range: Position
positions domain: Person, range: Position
receipts domain: Award or Honor, range: Award or Honor Receipt
selected publications domain: Organization, range: Authorship

From each of these pages, you can modify or delete the faux property that is displayed.

Alternatively, the Faux Property Editing form can be accessed by navigating to the Site Administration Page and clicking **Faux Property Listing**. This will display the complete list of faux properties. Click the title to access the editing form for that faux property.

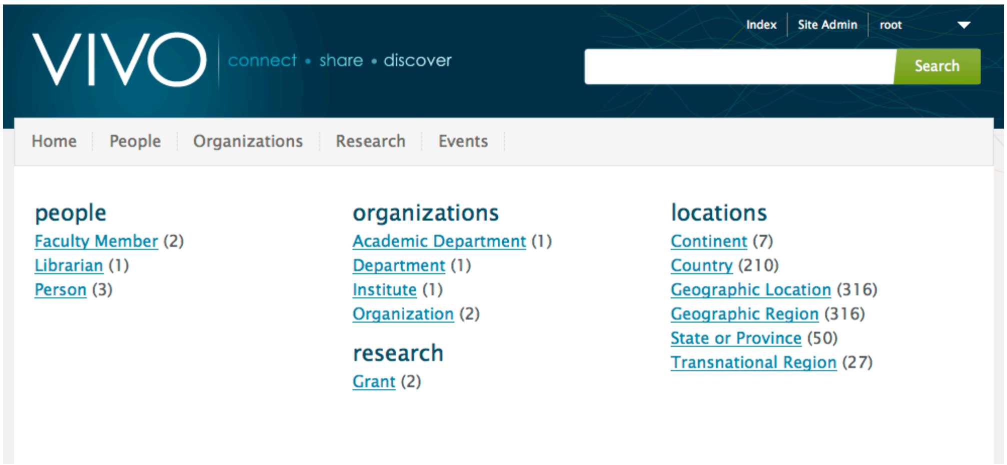
Note: Faux property custom list views are configured in the Faux Property Editing form, rather than as documented in Custom List View Configuration[111] for ontology properties.

8.3.6.4 Edit class groups

8.3.6.4.1 Overview

8.3.6.4.1.1 Before and After

Rename "people" to "personnel", and move it to the end of the display order.



people
Faculty Member (2)
Librarian (1)
Person (3)

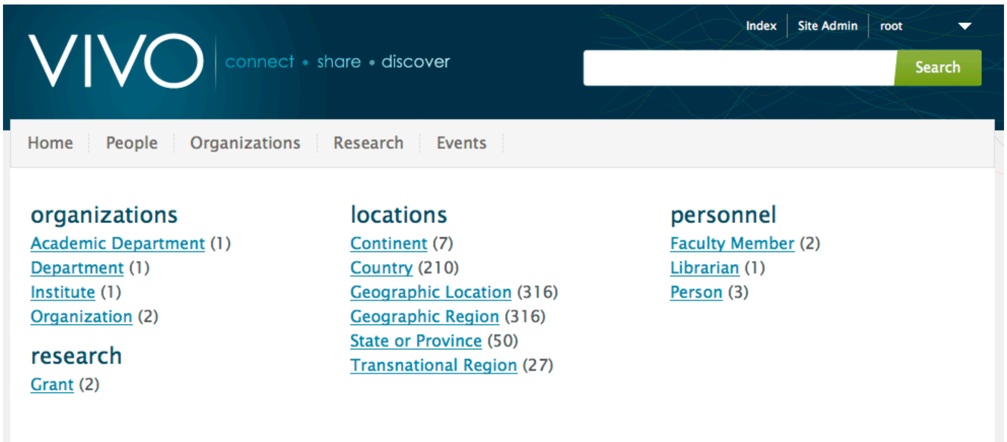
organizations
Academic Department (1)
Department (1)
Institute (1)
Organization (2)

research
Grant (2)

locations
Continent (7)
Country (210)
Geographic Location (316)
Geographic Region (316)
State or Province (50)
Transnational Region (27)

---

[111] https://wiki.duraspace.org/display/VIVO/Custom+List+View+Configurations


VIVO
connect • share • discover
Index
Site Admin
root
Search
Home
People
Organizations
Research
Events

organizations
Academic Department (1)
Department (1)
Institute (1)
Organization (2)

research
Grant (2)

locations
Continent (7)
Country (210)
Geographic Location (316)
Geographic Region (316)
State or Province (50)
Transnational Region (27)

personnel
Faculty Member (2)
Librarian (1)
Person (3)

8.3.6.4.2 What do you need to know?

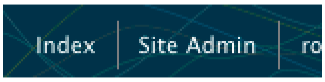
How to follow the GUI for class groups.

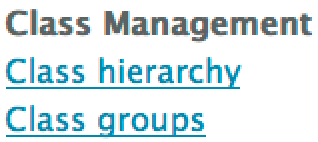
8.3.6.4.3 Getting started

VIVO comes with a default set of class groups. You can rename them, reorder them, create new groups or delete existing ones. You can also move classes from one group to another.

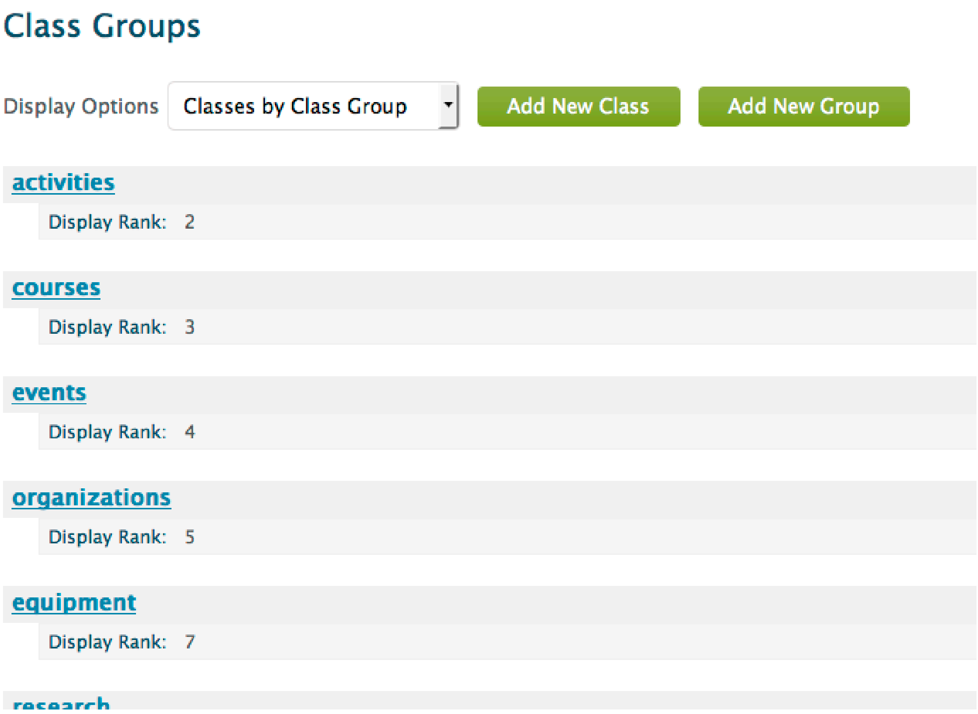
8.3.6.4.4 What to do

From the VIVO ***Site Admin*** page, navigate to the ***Class Groups*** page.


Index
Site Admin
ro


Class Management
Class hierarchy
Class groups

You can add a new class group, or click on the name of an existing group to edit it. You can also create a new class, but that involves editing the ontology.



Class Groups

Display Options: Classes by Class Group

activities
Display Rank: 2

courses
Display Rank: 3

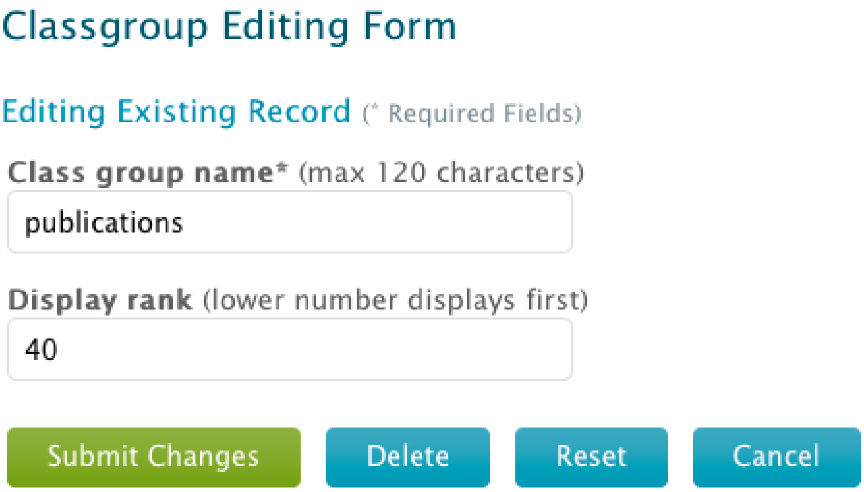
events
Display Rank: 4

organizations
Display Rank: 5

equipment
Display Rank: 7

research

You can change the name of a group, change its display rank, or delete it.



Classgroup Editing Form

Editing Existing Record (* Required Fields)

Class group name* (max 120 characters): publications

Display rank (lower number displays first): 40

When the index page is displayed, the class groups are shown in order of ascending display rank.

Classes that aren't included in any of the groups are not displayed on the index page. If you delete a class group, the classes that were in it will not be displayedon the index page unless you assign them to new groups.

Class groups and their membership can also affect the contents of managed pages. "Browse"-style pages display the contents of some or all of the classes in a single group, so the structure of your class groups will affect how these pages can be configured. Find more information in the section on Menu and page management (see page 163).

Class groups are also used to provide facets in search results.

8.3.6.5 Edit the appearance of classes

8.3.6.5.1 Overview

8.3.6.5.1.1 What can it do for you?

Change how VIVO displays a class of individuals.

For each class, you can change

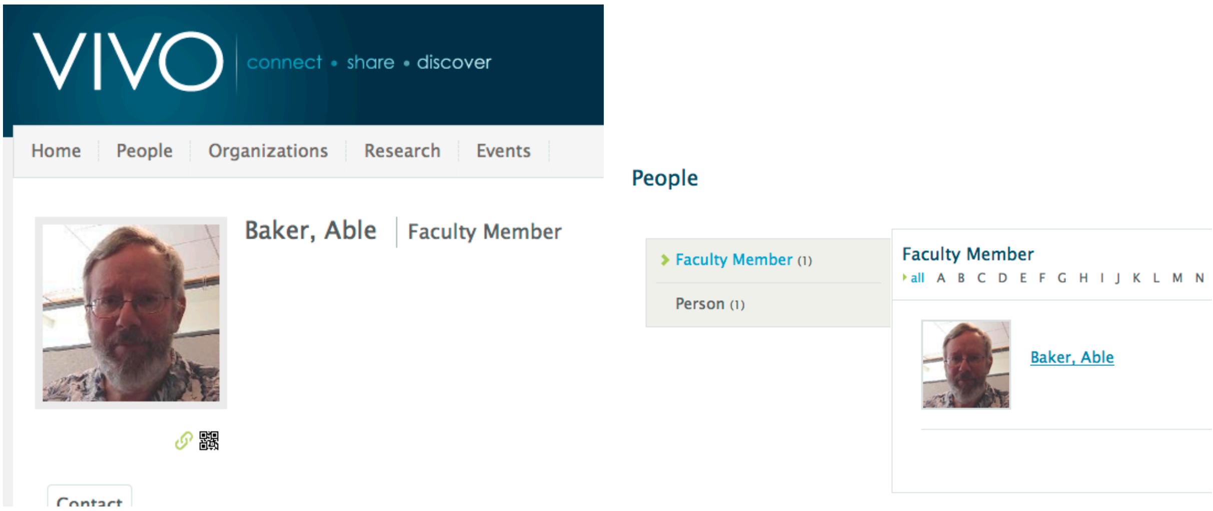
- the display label
- the public and private descriptions
- which class group it belongs to
- who can see the values
- who can edit the values
- whether the values will be published in linked open data requests

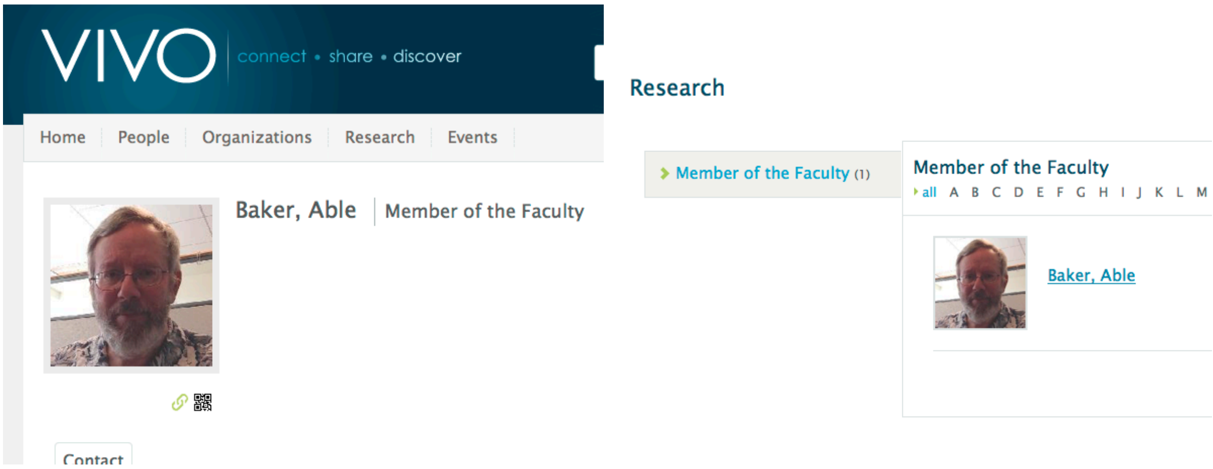
You can also change things like the namespace and parent class, but these are changes to the ontology.

The class editing form also allows you to assign a custom entry form to a class, as described in Custom entry forms (see page 217)

8.3.6.5.1.2 Before and After

Rename "Faculty Member" to "Member of the Faculty", and move it from the "People" class group to the "Research" class group.


VIVO
connect • share • discover
Home
People
Organizations
Research
Events
Baker, Able
Faculty Member
Contact
People
Faculty Member (1)
Person (1)
Faculty Member
all A B C D E F G H I J K L M N
Baker, Able


VIVO
connect • share • discover
Home
People
Organizations
Research
Events
Baker, Able
Member of the Faculty
Contact
Research
Member of the Faculty (1)
Member of the Faculty
all A B C D E F G H I J K L M
Baker, Able

8.3.6.5.1.3 What do you need to know?

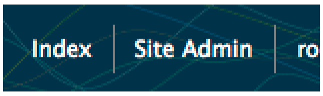
How to follow the GUI for class editing.—

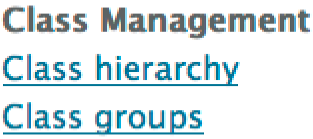
8.3.6.5.1.4 Getting started

VIVO comes with a default set of classes. You can edit them to suit your display requirements, or make more extensive modifications, by customizing the ontology.

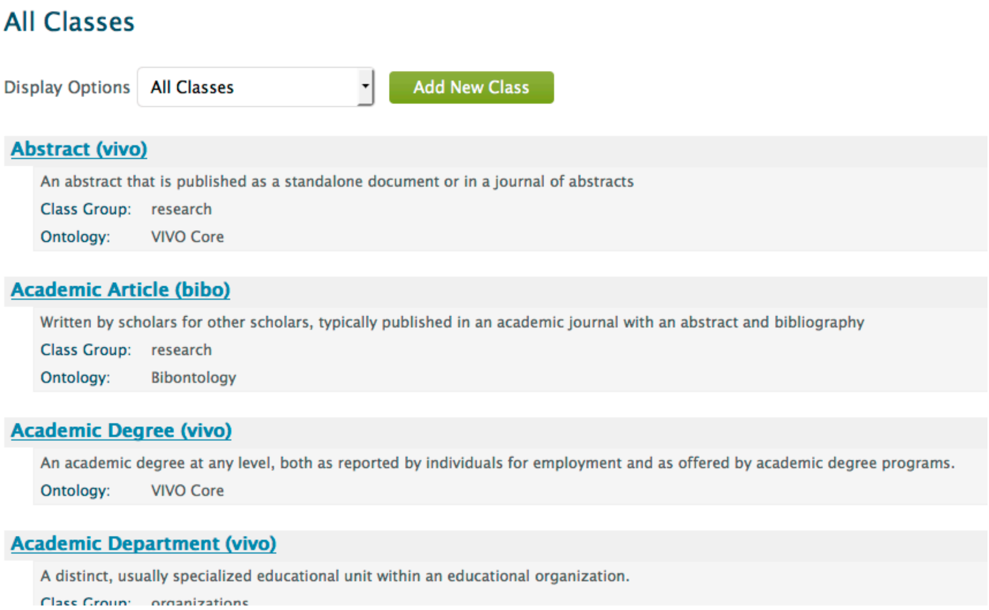
8.3.6.5.2 What to do

From the VIVO ***Site Admin*** page, navigate to the ***Class Hierarchy*** page.


Index
Site Admin
ro


Class Management
Class hierarchy
Class groups

If you know the ancestry of the class you want to change, you can navigate to it through the hierarchy. Otherwise, you may want to set the display options to show `All Classes`, and scroll directly to the class you want.



All Classes

Abstract (vivo)

An abstract that is published as a standalone document or in a journal of abstracts

Class Group: research

Ontology: VIVO Core

Academic Article (bibo)

Written by scholars for other scholars, typically published in an academic journal with an abstract and bibliography

Class Group: research

Ontology: Bibontology

Academic Degree (vivo)

An academic degree at any level, both as reported by individuals for employment and as offered by academic degree programs.

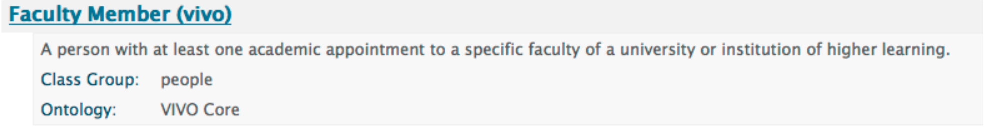
Ontology: VIVO Core

Academic Department (vivo)

A distinct, usually specialized educational unit within an educational organization.

Class Group: organizations

Click on the name of the class you want to edit.



Faculty Member (vivo)

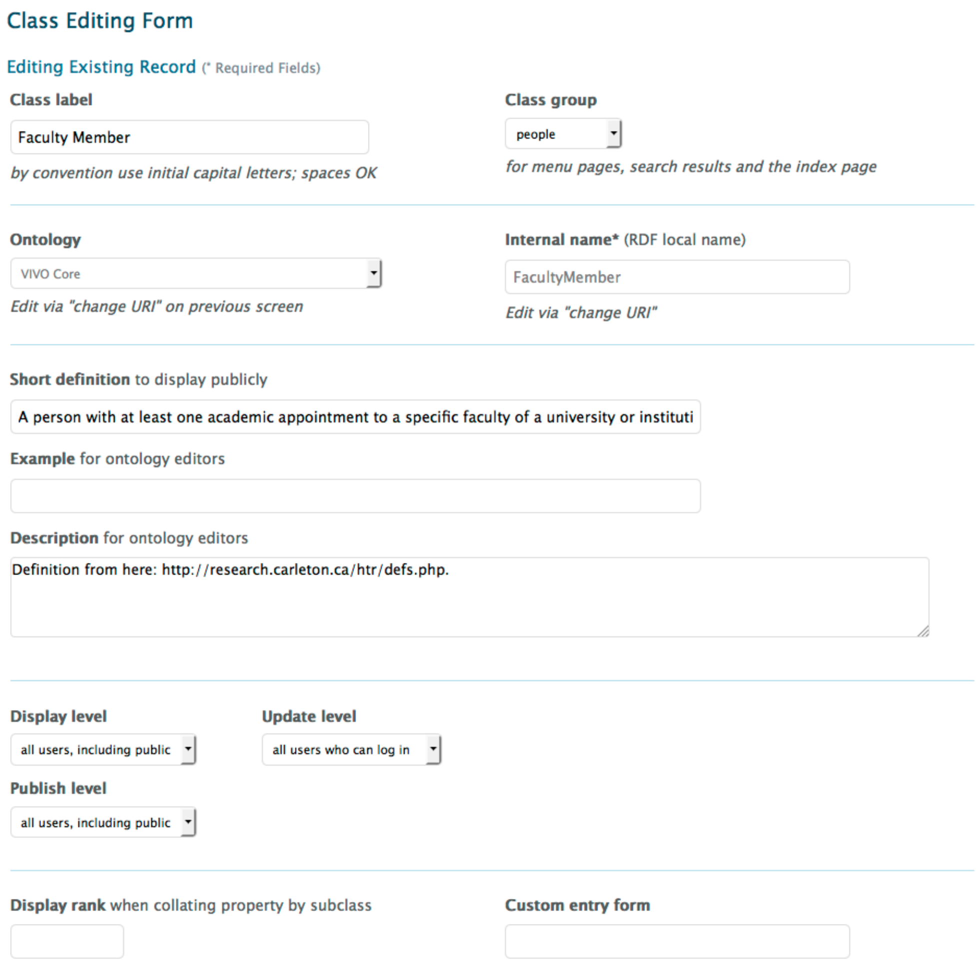
A person with at least one academic appointment to a specific faculty of a university or institution of higher learning.

Class Group: people

Ontology: VIVO Core

This shows you the ***Class Control Panel***. Click on the `Edit Class` button, and you will see the ***Class Editing* Form**.

You can change the class label, or the membership in a class group. You can also change the definition and description of the class.



Class Editing Form

Editing Existing Record (* Required Fields)

Class label: Faculty Member
by convention use initial capital letters; spaces OK

Class group: people
for menu pages, search results and the index page

Ontology: VIVO Core
Edit via "change URI" on previous screen

Internal name* (RDF local name): FacultyMember
Edit via "change URI"

Short definition to display publicly: A person with at least one academic appointment to a specific faculty of a university or instituti

Example for ontology editors:

Description for ontology editors: Definition from here: http://research.carleton.ca/htr/defs.php.

Display level: all users, including public

Update level: all users who can log in

Publish level: all users, including public

Display rank when collating property by subclass:

Custom entry form:

When you have made the changes, click on `Submit Changes`, and navigate to a page where you can see the results.

8.3.6.5.3 Notes

There is no need to restart or rebuild VIVO to see the effects of your changes.

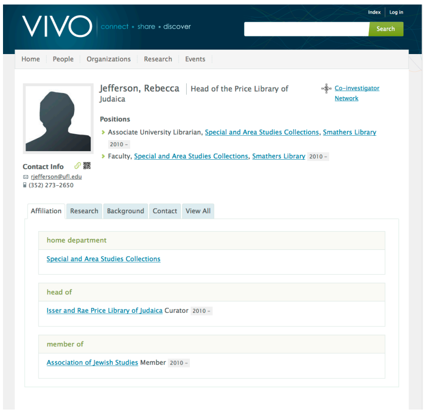
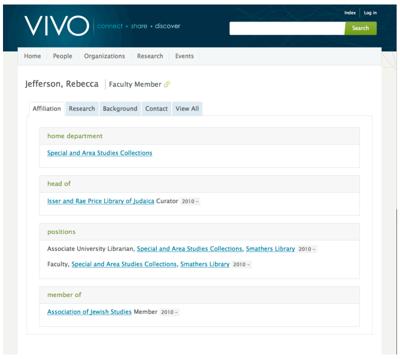
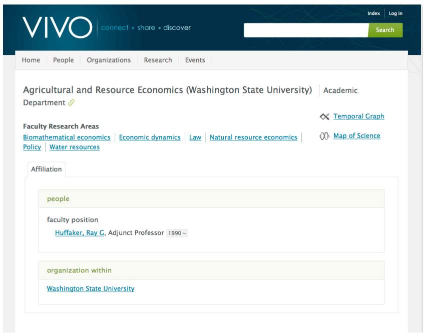
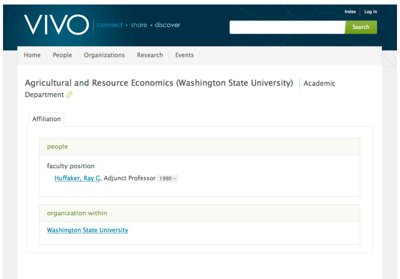
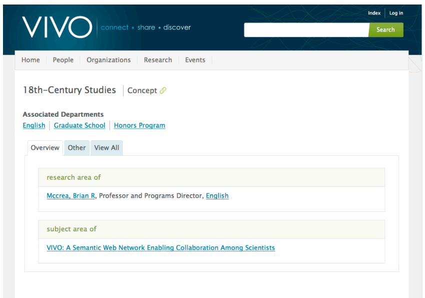
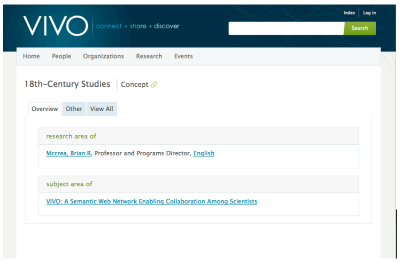
8.3.7 Class-specific templates for profile pages

8.3.7.1 Overview

When the profile page for an individual is created, it includes the standard header and footer, but most of the content is built by the Freemarker template `individual.ftl`.

Sometimes you would prefer for a particular class of individuals to have a particular style of profile page.For example, you want the contact information for a `foaf:Person` to appear right below their picture. Or you want to see a link to their co-investigator network near the top of the page.

VIVO lets you specify different templates for different classes of individuals. The standard distribution includes three of those specifications. Here are examples of profile pages for a Person, and Organization, and a Concept, with and without specified templates.

	specified template	**default template**
Person	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Affiliation Research Background Contact View All Jefferson, Rebecca | Head of the Price Library of Judaica Co-investigator Network **Positions** Associate University Librarian, Special and Area Studies Collections, Smathers Library 2010 - Faculty, Special and Area Studies Collections, Smathers Library 2010 - **Contact Info** rjefferson@ufl.edu (352) 273-2650 home department Special and Area Studies Collections head of Isser and Rae Price Library of Judaica Curator 2010 - member of Association of Jewish Studies Member 2010 -	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Affiliation Research Background Contact View All Jefferson, Rebecca | Faculty Member home department Special and Area Studies Collections head of Isser and Rae Price Library of Judaica Curator 2010 - positions Associate University Librarian, Special and Area Studies Collections, Smathers Library 2010 - Faculty, Special and Area Studies Collections, Smathers Library 2010 - member of Association of Jewish Studies Member 2010 -
Organization	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Affiliation Agricultural and Resource Economics (Washington State University) | Academic Department Temporal Graph Map of Science **Faculty Research Areas** Biomathematical economics | Economic dynamics | Law | Natural resource economics | Policy | Water resources people faculty position Huffaker, Ray G, Adjunct Professor 1990 - organization within Washington State University	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Affiliation Agricultural and Resource Economics (Washington State University) | Academic Department people faculty position Huffaker, Ray G, Adjunct Professor 1990 - organization within Washington State University
Concept	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Overview Other View All 18th-Century Studies | Concept **Associated Departments** English | Graduate School | Honors Program research area of Mccrea, Brian R, Professor and Programs Director, English subject area of VIVO: A Semantic Web Network Enabling Collaboration Among Scientists	 VIVO connect • share • discover Index Log in Search Home People Organizations Research Events Overview Other View All 18th-Century Studies | Concept research area of Mccrea, Brian R, Professor and Programs Director, English subject area of VIVO: A Semantic Web Network Enabling Collaboration Among Scientists

A specified profile template applies to individuals of the specified class, and to individuals of all sub-classes. So a page specified for `foaf:Person` will also apply to its sub-classes, like `vivo:FacultyMember` .

8.3.7.2 How to do it

There is no page in VIVO that will allow you to set or change these template specifications. Here are two ways to set it up.

8.3.7.2.1 Changes on an empty data model

When you start an empty VIVO instance for the first time, it will load the files in the `rdf/tbox/firsttime` directory into the `asserted-tbox` model.

The file `initialTBoxAnnotations.n3`, in this directory, contains the triples that specify these profile templates. They are scattered in the file, and mingled with other triples, but if you look, you can find these statements:

```
foaf:Organization
      vitro:customDisplayViewAnnot
              "individual--foaf-organization.ftl"^^xsd:string .
foaf:Person
      vitro:customDisplayViewAnnot
              "individual--foaf-person.ftl"^^xsd:string .
skos:Concept
      vitro:customDisplayViewAnnot
              "individual--skos-concept.ftl"^^xsd:string .
```

You can remove triples from this file prior to the first startup of VIVO, or add triples to create other template specifications.

8.3.7.2.2 Changes to an existing data model

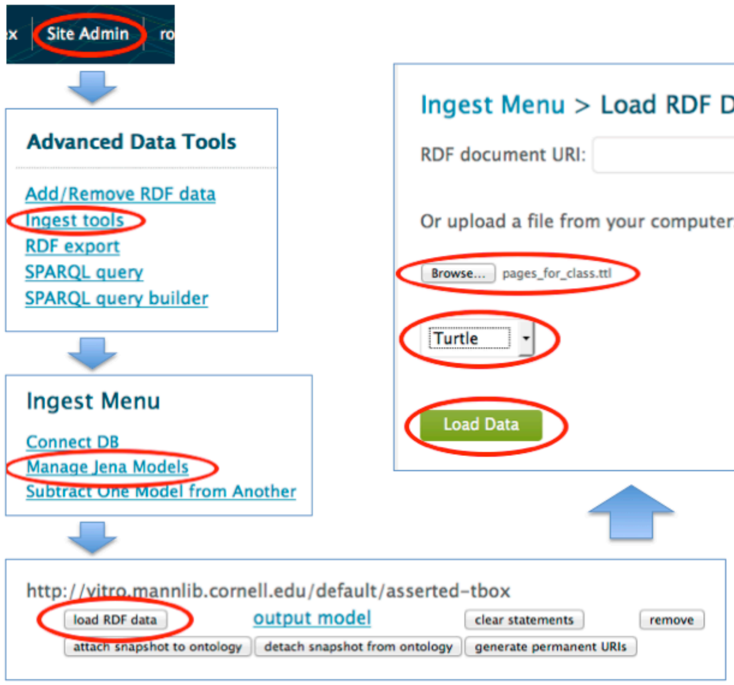
If VIVO has already been started, the files in `rdf/tbox/firsttime` will not be read again. You can use the advanced data tools on the Site Administrator's page to make changes.

8.3.7.2.2.1 Adding specifications

- Create a specialized Freemarker template.
- Prepare an RDF file containing the triples you want to add. Here is an example, in Turtle format:

```
@prefix bibo:    <http://purl.org/ontology/bibo/> .
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

bibo:Article
      vitro:customDisplayViewAnnot
              "individual--bibo-article.ftl"^^xsd:string .
```

- Login to VIVO as an admin user
- Follow the header link to the `Site Admin` page
- On the `Site Admin` page, under `Advanced Data Tools`, choose `Ingest tools`.
    - The link to `Add/Remove RDF data` is tempting, but it does not allow us to load into a specific model.
- On the `Ingest Menu` page, choose `Manage Jena Models`.
- In the list of available models, locate the controls for `http://vitro.mannlib.cornell.edu/default/asserted-tbox`, and choose `load RDF data`.
- On the `Load RDF Data` page, use the `Browse` control to locate your RDF file, and select the type of RDF format you used. Click the `Load Data` button.


Site Admin
Advanced Data Tools
Add/Remove RDF data
Ingest tools
RDF export
SPARQL query
SPARQL query builder
Ingest Menu
Connect DB
Manage Jena Models
Subtract One Model from Another
http://vitro.mannlib.cornell.edu/default/asserted-tbox
load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs
Ingest Menu > Load RDF Data
RDF document URI:
Or upload a file from your computer:
Browse...
pages_for_class.ttl
Turtle
Load Data

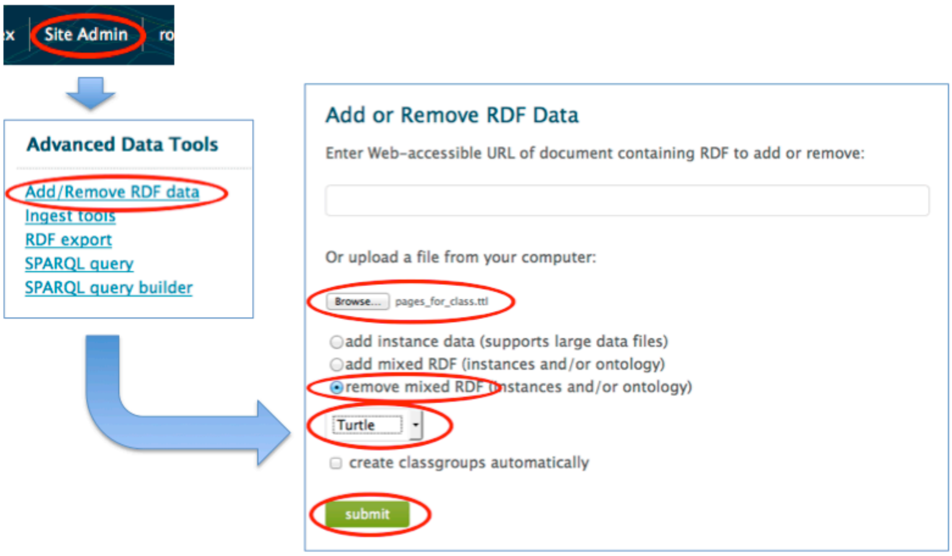
If there is a problem with the load, you will see a screen that shows an error message. Unfortunately, if the load is successful, you will see no indication. You will simply be returned to the list of available models.

You must restart VIVO to see the effect of your changes.

8.3.7.2.2.2 Removing specifications

- Create an RDF file containing the triples you want to remove. In this example, we will remove the triple that was added above, so we will use the same file.
- Login to VIVO as an admin user.
- Follow the header link to the `Site Admin` page.
- On the `Site Admin` page, under `Advanced Data Tools`, choose `Add/Remove RDF data`.

- On the `Add or Remove RDF Data` page, use the `Browse` control to locate your RDF file, choose to `remove mixed RDF`, and select the type of RDF format you used. Click the `submit` button.


Site Admin
Advanced Data Tools
Add/Remove RDF data
Ingest tools
RDF export
SPARQL query
SPARQL query builder

Add or Remove RDF Data

Enter Web-accessible URL of document containing RDF to add or remove:

Or upload a file from your computer:

Browse... pages_for_class.ttl

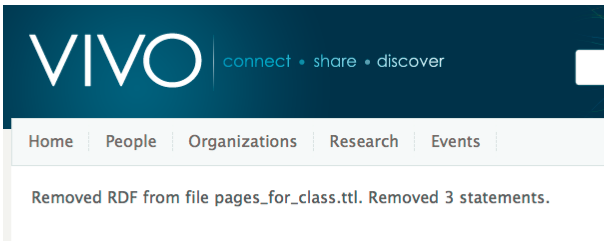
○ add instance data (supports large data files)
○ add mixed RDF (instances and/or ontology)
● remove mixed RDF (instances and/or ontology)

Turtle

☐ create classgroups automatically

submit

If there is a problem with your data file, you will see a screen showing an error message. If the removal is successful, you will see a message like the following:


VIVO
connect • share • discover
Home
People
Organizations
Research
Events

Removed RDF from file pages_for_class.ttl. Removed 3 statements.

Note that the message tells you how many triples you asked to have removed, without regard to whether they actually existed in the data model.

You must restart VIVO to see the effect of your changes.

8.3.7.2.2.3 Changing specifications

There is no direct way to replace triples in the data model. Use the preceding steps to remove the triples you don't want, and to add new triples to replace them.

8.3.7.2.3 Other mechanisms

Expert VIVO users will be aware of many other ways of adding or removing triples.

Remember that VIVO must be restarted, since the list of specific templates is created at startup.

8.3.8 Excluding Classes from the Search

8.3.8.1 Overview

A VIVO/Vitro instance can be customized to exclude a class of individuals from the search index. All instances of a class can be excluded from the search index by adding a vitroDisplay:excludeClass[112] property between vitroDisplay:SearchIndex and the URI of the class that you would like to exclude. This will have the effect of not displaying any individual with this class in search results, on the index page, in browse pages and as options for entry in some forms. The search exclusions are controlled by RDF statements in the display model.

8.3.8.2 Steps to create a new search exclusion

1. Create a file with your new exclusion
2. In your deploy directories, place that file in either `vivo/rdf/display/everytime` or `vitro/webapp/rdf/display/everytime`
3. Stop Tomcat, deploy, and restart Tomcat
4. Login to VIVO as an admin and rebuild the search index.

8.3.8.3 Example on the contents of an RDF file to define exclusions

```
@prefix vitroDisplay: <http://vitro.mannlib.cornell.edu/ontologies/display/1.1#> .

vitroDisplay:SearchIndex
    vitroDisplay:excludeClass <http://example.org/ns/ex/Hat> .
```

8.3.9 Custom List View Configurations

8.3.9.1 Introduction

Custom list views provide a way to expand the data that is displayed for object and data properties. For example, with the default list view the hasPresenterRole object property would only display the rdfs:label of the role individual; but with a custom list view, the "presentations" view includes not only the role but also the title of the presentation, the name of the conference where the presentation was given and the date the presentation was given. This wiki page provides guidelines for developing custom list views as well as an example of a custom list view.

---

[112] http://vitroDisplayexcludeClass

8.3.9.2 List View Configuration Guidelines

8.3.9.2.1 Registering the List View

A custom list view is associated with an object property in the RDF files in the directory `/vivo/rdf/display/everytime`. To register a list view, create a new `.rdf` or `.n3` file in that directory. The file must be well-formed RDF/XML or N3.

Here is an example of registering a new association in a file named `newListViews.n3`:

```
<http://vivoweb.org/ontology/core#authorInAuthorship>
<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#listViewConfigFile>
"listViewConfig-authorInAuthorship.xml" .
```

With this triple the `authorInAuthorship` object property is associated with a list view configuration that is defined in a file named `listViewConfig-authorInAuthorship.xml`.

Place the N3 file containing this triple (or the well-formed RDF/XML file) in the `/vivo/rdf/display/everytime` directory, redeploy VIVO and restart tomcat to put the new custom list view in effect.

Note: Faux property custom list views are not registered in the same way. The list view is specified in the configuration of the faux property itself, using the faux property editing form. See details in Create and edit faux properties. (see page 172)

8.3.9.2.2 Required Elements

The list view configuration file requires three elements:

1. list-view-config: this is the root element that contains the other elements
2. query-select: this defines the SPARQL query used to retrieve data
3. template: this holds the name of the Freemarker template file used to display a single property statement

8.3.9.2.3 Optional Elements

The list-view-config root element can also contain two optional elements:

1. query-construct: one or more construct queries used to construct a model that the select query is run against
2. postprocessor: a Java class that postprocesses the data retrieved from the query before sending it to the template. If no post-processor is specified, the default post-processor will be invoked.

8.3.9.2.4 Construct Queries

Because SPARQL queries with multiple OPTIONAL clauses are converted to highly inefficient SQL by the Jena API, one or more construct queries should be included to improve query performance. They are used to construct a model significantly smaller than the entire dataset that the select query can be run against with reasonable performance.

The construct queries themselves should not contain multiple OPTIONAL clauses, to prevent the same type of inefficiency. Instead, use multiple construct queries to construct a model that includes all the necessary data.

In the absence of any construct queries, the select query is run against the entire dataset. If your select query does not involve a lot of OPTIONAL clauses, you do not need to include construct queries.

The construct queries must be designed to collect all the data that the select query will request. They can be flexibly constructed to contain more data than is currently selected, to allow for possible future expansion of the SELECT and to simplify the WHERE clause. For example, one of the construct queries for core:hasRole[113] includes:

```
CONSTRUCT {
    ?role ?roleProperty ?roleValue .
    ...
} WHERE {
    ?role ?roleProperty ?roleValue .
    ...
}
```

That is, it includes all the properties of the role, rather than just those currently selected by the select query.

The ordering of the construct queries is not significant.

8.3.9.2.5 The Select Query

8.3.9.2.5.1 General select query requirements

Use a SELECT DISTINCT clause rather than a simple SELECT. There can still be cases where the same individual is retrieved more than once, if there are multiple solutions to the other assertions, but DISTINCT provides a start at uniqueness.

The WHERE clause must contain a statement ?subject ?property ?object, with the variables ?subject and ?property named as such. For a default list view, the ?object variable must also be named as such. For a custom list view, the object can be given any name, but it must be included in the SELECT terms retrieved by the query. This is the statement that will be edited from the edit links.

8.3.9.2.5.2 Data which is required in public view, optional when editing

Incomplete data can result in a missing linked individual or other critical data (such as a URL or anchor text on a link object). When the user has editing privileges on the page, these statements are displayed so that the user can edit them and provide the missing data. They should be hidden from non-editors. Follow these steps in the select query to ensure this behavior:

---

[113] http://corehasRole

- Enclose the clause for the linked individual in an OPTIONAL block.
- Select the object's localname using the ARQ localname function, so that the template can display the local name in the absence of the linked individual. Alternatively, this can be retrieved in the template using the localname(uri) method.
- Require the optional information in the public view by adding a filter clause which ensures that the variable has been bound, inside tag <critical-data-required>. For example:

```
OPTIONAL { ?authorship core:linkedInformationResource ?infoResource }
```

- This statement is optional because when editing we want to display an authorship that is missing a link to an information resource. Then add:

```
<critical-data-required>
    FILTER ( bound(?infoResource) )
</critical-data-required>
```

- The Java code will preprocess the query to remove the <critical-data-required> tag, either retaining its text content (in public view) or removing the content (when editing), so that the appropriate query is executed.

8.3.9.2.5.3 Collated vs. uncollated queries

The query should contain <collated> elements, which are used when the property is collated. For uncollated queries, the fragments are removed by a query preprocessor. Since any ontology property can be collated in the Ontology Editor, all queries should contain the following <collated> elements:

- A ?subclass variable, named as such, in the SELECT clause. If the ?subclass variable is missing, the property will be displayed without collation.

```
SELECT DISTINCT <collated> ?subclass </collated> ...
```

- ?subclass must be the first term in the ORDER BY clause.

```
ORDER BY <collated> ?subclass </collated> ...
```

- Include the following in the WHERE clause, substituting in the relevant variables for ?infoResource and core:InformationResource:

```
<collated>
    OPTIONAL { ?infoResource a ?subclass
               ?subclass rdfs:subClassOf core:InformationResource .
    }
</collated>
```

Postprocessing removes all but the most specific subclass value from the query result set.

Alternatively (and preferably):

```
<collated>
    OPTIONAL { ?infoResource vitro:mostSpecificType ?subclass
               ?subclass rdfs:subClassOf core:InformationResource .
    }
</collated>
```

Automatic postprocessing to filter out all but the most specific subclass will be removed in favor of this implementation in the future.

Both collated and uncollated versions of the query should be tested, since the collation value is user-configurable via the ontology editor.

8.3.9.2.5.4 Datetimes in the query

To retrieve a datetime interval, use the following fragment, substituting the appropriate variable for ?edTraining:

```
OPTIONAL {
    ?edTraining core:dateTimeInterval ?dateTimeInterval
    OPTIONAL { ?dateTimeInterval core:start ?dateTimeStartValue .
               ?dateTimeStartValue core:dateTime ?dateTimeStart
    }
    OPTIONAL { ?dateTimeInterval core:end ?dateTimeEndValue .
               ?dateTimeEndValue core:dateTime ?dateTimeEnd
    }
}
```

The variables ?dateTimeStart and ?dateTimeEnd are included in the SELECT clause.

Many properties that retrieve dates order by end datetime descending (most recent first). In this case, a post-processor must apply to sort null values at the top rather than the bottom of the list, which is the ordering returned by the SPARQL ORDER BY clause in the case of nulls in a descending order. In that case, the variable names must be exactly as shown to allow the post-processor to do its work.

8.3.9.2.6 The Template

To ensure that values set in the template on one iteration do not bleed into the next statement:

- The template should consist of a macro that controls the display, and a single line that invokes the macro.
- Variables defined inside the macro should be defined with <#local> rather than <#assign>.

To allow for a missing linked individual, the template should include code such as:

```
<#local linkedIndividual>
    <#if statement.org??>
        <a href="${url(statement.org)}">${statement.orgName}</a>
    <#else>
        <#-- This shouldn't happen, but we must provide for it -->
        <a href="${url(statement.edTraining)}">${statement.edTrainingName}</a> (no
linked organization)
    </#if>
</#local>
```

The query must have been constructed to return orgName (see above under "General select query requirements"), or alternatively the template can use the localname function: ${localname(org)}.

If a variable is in an OPTIONAL clause in the query, the display of the value in the template should include the default value operator ! to prevent an error on null values.

8.3.9.3 List View Example

This example will walk through the custom list view for the core:researchAreaOf object property. This property is displayed on the profile page for research area individuals.

8.3.9.3.1 Associate the property with a list view

In this example we're using RDF/XML to associate the researchAreaOf object property (line 1) with a specific list view configuration (line 2):

```
<rdf:Description rdf:about="http://vivoweb.org/ontology/core#researchAreaOf">
    <display:listViewConfigFile rdf:datatype="http://www.w3.org/2001/
XMLSchema#string">listViewConfig-researchAreaOf.xml</display:listViewConfigFile>
</rdf:Description>
```

8.3.9.3.2 The list view configuration

The root <list-view-config> element in our listViewConfig-researchAreaOf.xml file contains the required <query-select> and <template> elements as well as two optional <query-construct> sections and an optional <postprocessor> element.

This is the <query-select> element:

```
    <query-select>
        PREFIX afn:  &lt;http://jena.hpl.hp.com/ARQ/function#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX vitro: &lt;http://vitro.mannlib.cornell.edu/ns/vitro/0.7#&gt;
        PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt;

        SELECT DISTINCT
                       ?person
                       ?personName
                       ?posnLabel
                       ?orgLabel
                       ?type
                       ?personType
                       ?title
        WHERE {
                 ?subject ?property ?person .
                 ?person core:personInPosition ?position .
                 OPTIONAL { ?person rdfs:label ?personName }
                 OPTIONAL { ?person core:preferredTitle ?title }
                 OPTIONAL { ?person vitro:mostSpecificType ?personType .
                            ?personType rdfs:subClassOf foaf:Person
                 }
                 OPTIONAL { ?position rdfs:label ?posnLabel }
                 OPTIONAL { ?position core:positionInOrganization ?org .
                            ?org rdfs:label ?orgLabel
                 }
                 OPTIONAL { ?position core:hrJobTitle ?hrJobTitle }
                 OPTIONAL { ?position core:rank ?rank }
        }
        ORDER BY ?personName ?type
    </query-select>
```

Here is the first <query-construct> element:

```
    <query-construct>
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;

        CONSTRUCT {
            ?subject ?property ?person .
            ?person core:personInPosition ?position .
            ?position rdfs:label ?positionLabel .
            ?position core:positionInOrganization ?org .
            ?org rdfs:label ?orgName .
            ?position core:hrJobTitle ?hrJobTitle
        } WHERE {
            {
                ?subject ?property ?person
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position rdfs:label ?positionLabel
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:positionInOrganization ?org
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:positionInOrganization ?org .
                ?org rdfs:label ?orgName
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:hrJobTitle ?hrJobTitle
            }
        }
    </query-construct>
```

The second <query-construct> element:

```
    <query-construct>
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;
        PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt;
        PREFIX vitro: &lt;http://vitro.mannlib.cornell.edu/ns/vitro/0.7#&gt;

        CONSTRUCT {
            ?subject ?property ?person .
            ?person rdfs:label ?label .
            ?person core:preferredTitle ?title .
            ?person vitro:mostSpecificType ?personType .
            ?personType rdfs:subClassOf foaf:Person
        } WHERE {
            {
                ?subject ?property ?person
            } UNION {
                ?subject ?property ?person .
                ?person rdfs:label ?label
            } UNION {
                ?subject ?property ?person .
                ?person core:preferredTitle ?title
            }  UNION {
                ?subject ?property ?person .
                ?person vitro:mostSpecificType ?personType .
                ?personType rdfs:subClassOf foaf:Person
            }
        }
    </query-construct>
```

Next is the required <template> element:

```
<template>propStatement-researchAreaOf.ftl</template>
```

And here is the <postprocessor> element:

```
<postprocessor>edu.cornell.mannlib.vitro.webapp.web.templatemodels.individual.Researc
hAreaOfPostProcessor</postprocessor>
```

Note: the <postprocessor> is included here only to show the syntax. The actual listViewConfig-researchAreaOf.xml file in the VIVO code base does not use a custom post-processor.

8.3.9.3.3 The Freemarker Template

Finally, here are the contents of our Freemarker template, propStatement-researchAreaOf.ftl.

```
<#import "lib-sequence.ftl" as s>
<@showResearchers statement />
<#-- Use a macro to keep variable assignments local; otherwise the values carry over
to the
     next statement -->
<#macro showResearchers statement>
    <#local linkedIndividual>
        <a href="${profileUrl(statement.uri("person"))}" title="$
{i18n().person_name}">${statement.personName}</a>
    </#local>
    <#if statement.title?has_content >
        <#local posnTitle = statement.title>
    <#else>
        <#local posnTitle = statement.posnLabel!statement.personType>
    </#if>
    <@s.join [ linkedIndividual, posnTitle, statement.orgLabel!"" ] /> $
{statement.type!}
</#macro>
```

8.3.10 Creating short views of individuals

8.3.10.1 Overview

8.3.10.1.1 What does it do?

Custom short views are used in three different contexts within VIVO, to give you more control over how an individual is displayed in that context.

You can configure VIVO to display different classes of individuals in different ways. As an example, you might choose to display a Faculty Member in a grey color if she has no current appointments.

Custom short views will frequently be different in the three different contexts in which they are available. For example, you might want to show the image of a Person on a search result, but you might not want to display that image in the Person index pages.

8.3.10.1.2 How is it created?

A short view is defined by two elements. First there will be a group of RDF statements in this file:

`vitro/webapp/web/WEB-INF/resources/shortview_config.n3`

In a VIVO release, this file is empty (except for a few comments), and the default short views are used in all cases. You will add RDF to this file as you define your custom short views. The RDF statements will name the class of Individual, the Freemarker template, and any SPARQL queries that are used to get the data you need to display.

The other thing you will need is the Freemarker template itself, to render your custom view.

An example of some custom short views can be found in this directory:

`vivo/utilities/acceptance-tests/suites/ShortViews/`

The directory contains a copy of `shortview_config.n3`, and some Freemarker templates. These files are essentially the same as the examples on this page.

8.3.10.2 Details

8.3.10.2.1 The class association

A statement associates a custom short view with a VIVO Class from the ontology. For example:

```
vivo:FacultyMember
    display:hasCustomView   mydomain:facultySearchView .
```

This means that the specified short view will be used for any Individual that has `vivo:FacultyMember` as a most specific class.

Only the most specific classes of an Individual are recognized by the short views. So if you want a custom short view to be used for all Person objects, you must define it on Person and on FacultyMember and on FacultyMemberEmeritus, etc.

8.3.10.2.2 The `customViewForIndividual` definition

An object is given a URI and declared to be a `customViewForIndividual`

It may apply to one or more of the contexts: `SEARCH, INDEX`, or `BROWSE`.
It must have an associated template, and may have one or more associated DataGetters.

Here is an example:

```
mydomain:facultySearchView
    a                        display:customViewForIndividual ;
    display:appliesToContext "SEARCH" ;
    display:hasTemplate      "view-search-faculty.ftl" ;
    display:hasDataGetter    mydomain:facultyDepartmentDG .
```

This custom view applies in the `SEARCH` context. It specifies a DataGetter, which will be invoked to find data each time this short view is rendered. It also specifies the Freemarker template that will render the view.

8.3.10.2.3 The `SparqlQueryDataGetter` definition

The DataGetter must also be defined in the RDF, like this:

```
mydomain:facultyDepartmentDG
    a                   datagetters:SparqlQueryDataGetter ;
    display:saveToVar   "details" ;
    display:query       """
       PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
       PREFIX vivo: <http://vivoweb.org/ontology/core#>
       SELECT ?deptName
       WHERE {
          ?individualUri  vivo:hasMemberRole      ?membership .
          ?membership     vivo:roleContributesTo  ?deptUri .
          ?deptUri
             a           vivo:AcademicDepartment ;
             rdfs:label  ?deptName .
       }
       LIMIT 20
       """ .
```

Besides the type and the URI, this object specifies a SPARQL query, and the name of a Freemarker variable where the results of the query will be stored.

When the SPARQL query is executed, the value of `?individualUrl` will be bound to the actual URI of the Individual being displayed. The values returned from the query will be stored in an array of Freemarker Hash containers, with each one representing a row of the SPARQL query result. The Hash will contain the values returned by the query, keyed to the variable names used in the query.

When this array of Hash containers has been constructed, it is stored in the variable named by the DataGetter declaration; `details` in this case.

The DataGetter is optional in a custom short view. If no DataGetter is specified, then the Freemarker template will only have the standard set of data available to it.

Conversely, multiple DataGetters may be specified on a short view. If this is done, each DataGetter should assign to a different Freemarker variable, to avoid problems with overwriting data.

8.3.10.2.4 The Freemarker template

A default template exists for each of the short view contexts: `SEARCH` , `INDEX` and `BROWSE` .
If no custom short view is defined for an Individual, the default template will be used to render the Individual.

The custom template will likely be based on the default template for that context. For example, the default template for search results is called `view-search-default.ftl` and looks like this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<p class="snippet">${individual.snippet}</p>
```

Our modified template is this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<#if (details[0].deptName)?? >
    <span class="display-title">Member of ${details[0].deptName}</span>
</#if>

<p class="snippet">${individual.snippet}</p>
```

So, if a department name was found for this Faculty member, it will be displayed. If more than one was found, the remainder will be ignored, since the template only displays the first one.

8.3.10.2.5 The default template can be modified in the theme

Besides taking advantage of custom short views, the theme author may also choose to override the templates for the default short views. This would merely require creating a new template with the same name as the one being overridden, and putting this new template into the template directory of your theme.

8.3.10.3 Some examples

8.3.10.3.1 The scenario

When a FacultyMember appears in a short view, we would like to add the name of his/her department. This information isn't directly available, so we will need to obtain it from a SPARQL query.

This should work in all three contexts, `SEARCH`, `INDEX`, and `BROWSE`.

This will only apply to FacultyMembers. Other individuals will use the default short views.

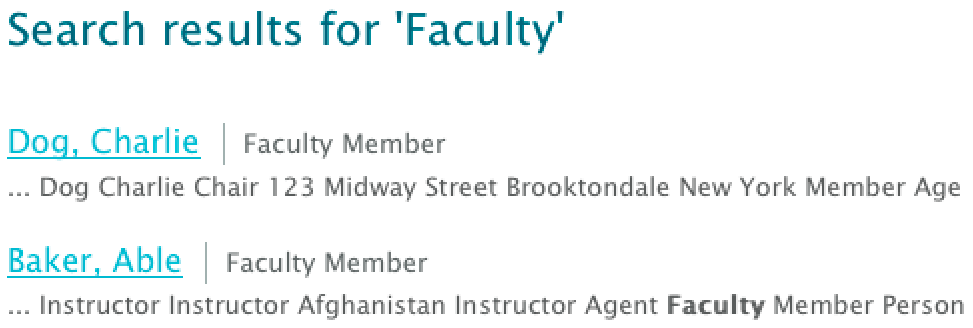
If the FacultyMember is not a member of a department, the short view should just omit the name, without causing an error.

8.3.10.3.2 SEARCH example

8.3.10.3.2.1 The default template

The default short view for the SEARCH context looks like this:

And it produces results like this:



Search results for 'Faculty'

Dog, Charlie | Faculty Member
... Dog Charlie Chair 123 Midway Street Brooktondale New York Member Age

Baker, Able | Faculty Member
... Instructor Instructor Afghanistan Instructor Agent **Faculty** Member Person

8.3.10.3.2.2 Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

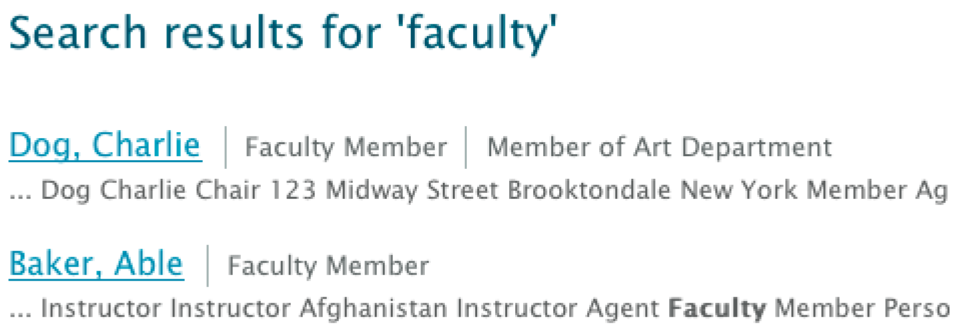
```
vivo:FacultyMember
    display:hasCustomView   mydomain:facultySearchView .

mydomain:facultySearchView
    a                         display:customViewForIndividual ;
    display:appliesToContext  "SEARCH" ;
    display:hasTemplate       "view-search-faculty.ftl" ;
    display:hasDataGetter     mydomain:facultyDepartmentDG .

mydomain:facultyDepartmentDG
    a                  datagetters:SparqlQueryDataGetter ;
    display:saveToVar  "details" ;
    display:query      """
       PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
       PREFIX vivo: <http://vivoweb.org/ontology/core#>
       SELECT ?deptName
       WHERE {
          ?individualUri  vivo:hasMemberRole      ?membership .
          ?membership     vivo:roleContributesTo  ?deptUri .
```

```
            ?deptUri
               a              vivo:AcademicDepartment ;
               rdfs:label     ?deptName .
         }
         LIMIT 20
         """ .
```

Create the template `view-search-faculty.ftl` to look like this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<#if (details[0].deptName)?? >
    <span class="display-title">Member of ${details[0].deptName}</span>
</#if>

<p class="snippet">${individual.snippet}</p>
```

The new search results look like this:



Search results for 'faculty'

Dog, Charlie | Faculty Member | Member of Art Department
... Dog Charlie Chair 123 Midway Street Brooktondale New York Member Ag

Baker, Able | Faculty Member
... Instructor Instructor Afghanistan Instructor Agent **Faculty** Member Perso

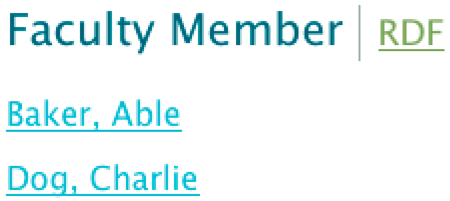
8.3.10.3.3 INDEX example

8.3.10.3.3.1 The default template

The default short view for the `INDEX` context looks like this:

```
<#import "lib-properties.ftl" as p>
<a href="${individual.profileUrl}" title="name">${individual.name}</a>
<@p.mostSpecificTypes individual />
```

And it produces results like this:



Faculty Member | RDF

Baker, Able

Dog, Charlie

8.3.10.3.3.2 Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

```
vivo:FacultyMember
    display:hasCustomView   mydomain:facultyIndexView .

mydomain:facultyIndexView
    a                          display:customViewForIndividual ;
    display:appliesToContext   "INDEX" ;
    display:hasTemplate        "view-index-faculty.ftl" ;
    display:hasDataGetter      mydomain:facultyDepartmentDG .

mydomain:facultyDepartmentDG
    a                   datagetters:SparqlQueryDataGetter ;
    display:saveToVar   "details" ;
    display:query       """
       PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
       PREFIX vivo: <http://vivoweb.org/ontology/core#>
       SELECT ?deptName
       WHERE {
          ?individualUri  vivo:hasMemberRole      ?membership .
          ?membership     vivo:roleContributesTo  ?deptUri .
          ?deptUri
             a            vivo:AcademicDepartment ;
             rdfs:label   ?deptName .
       }
       LIMIT 20
       """ .
```

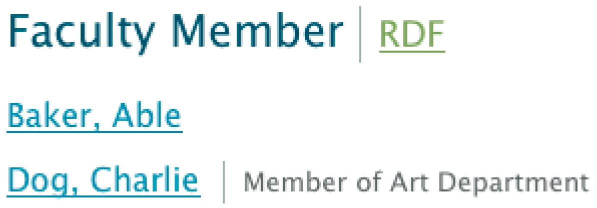
Note that the DataGetter is the same as in the previous example. If two custom short views want to use the same DataGetter, there is no need to code it twice. If both of these examples are tried at the same time, the two custom short views would refer to the same DataGetter.

Create the template `view-index-faculty.ftl` to look like this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>
```

```
<@p.displayTitle individual />

<#if (details[0].deptName)?? >
    <span class="display-title">Member of ${details[0].deptName}</span>
</#if>
```

The new index looks like this:



Faculty Member | RDF

Baker, Able

Dog, Charlie | Member of Art Department

8.3.10.3.4 BROWSE example

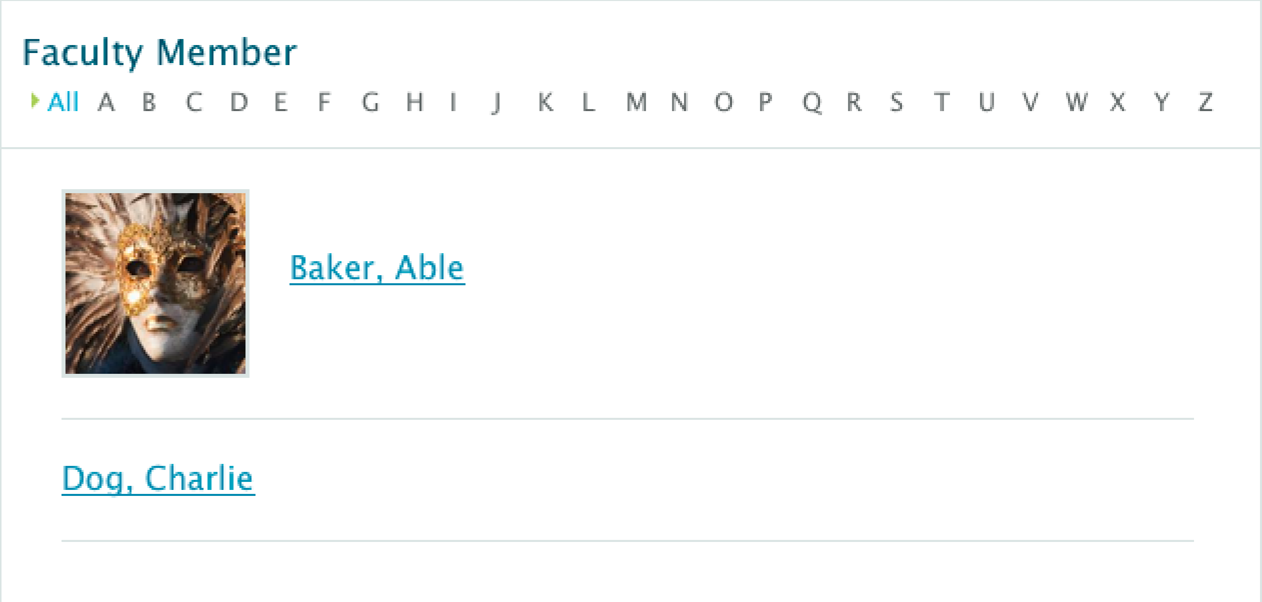
8.3.10.3.4.1 The default template

The default short view for the `BROWSE` context looks like this:

```
<#import "lib-properties.ftl" as p>

<li class="individual" role="listitem" role="navigation">

<#if (individual.thumbUrl)??>
    <img src="${individual.thumbUrl}" width="90" alt="${individual.name}" />
    <h1 class="thumb">
        <a href="${individual.profileUrl}" title="${i18n().view_profile_page_for}
              ${individual.name}}">${individual.name}</a>
    </h1>
<#else>
    <h1>
        <a href="${individual.profileUrl}" title="${i18n().view_profile_page_for}
              ${individual.name}}">${individual.name}</a>
    </h1>
</#if>

<#assign cleanTypes =
    'edu.cornell.mannlib.vitro.webapp.web.TemplateUtils$DropFromSequence'?new()
(individual.mostSpecificTypes, vclass) />
<#if cleanTypes?size == 1>
    <span class="title">${cleanTypes[0]}</span>
<#elseif (cleanTypes?size > 1) >
    <span class="title">
        <ul>
            <#list cleanTypes as type>
            <li>${type}</li>
            </#list>
        </ul>
    </span>
</#if>
</li>
```

Notice that it contains some conditional logic, producing different results depending on whether there is an image file associated with the Individual, or whether the type being browsed is the most specific type for the individual.

The default template produces results like this:



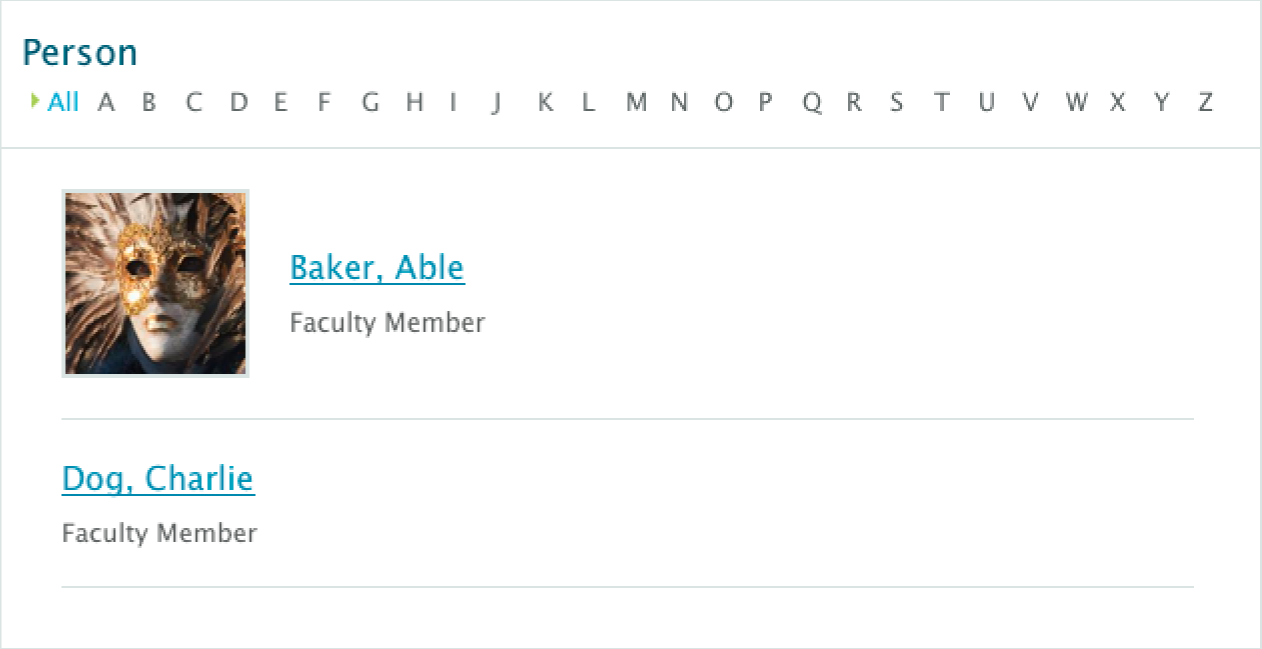
Faculty Member

All A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Baker, Able

Dog, Charlie

Or this:



Person

All A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Baker, Able
Faculty Member

Dog, Charlie
Faculty Member

8.3.10.3.4.2 Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

```
vivo:FacultyMember
    display:hasCustomView   mydomain:facultyBrowseView .

mydomain:facultyBrowseView
    a                          display:customViewForIndividual ;
    display:appliesToContext   "BROWSE" ;
    display:hasTemplate        "view-browse-faculty.ftl" ;
    display:hasDataGetter      mydomain:facultyDepartmentDG ;
    display:hasDataGetter      mydomain:facultyPreferredTitleDG .

mydomain:facultyDepartmentDG
    a                   datagetters:SparqlQueryDataGetter ;
    display:saveToVar   "details" ;
    display:query       """
       PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
       PREFIX vivo: <http://vivoweb.org/ontology/core#>
       SELECT ?deptName
       WHERE {
          ?individualUri  vivo:hasMemberRole      ?membership .
          ?membership     vivo:roleContributesTo  ?deptUri .
          ?deptUri
             a            vivo:AcademicDepartment ;
             rdfs:label   ?deptName .
       }
       LIMIT 20
       """ .

mydomain:facultyPreferredTitleDG
    a                  datagetters:SparqlQueryDataGetter ;
    display:saveToVar  "extra" ;
    display:query      """
       PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
       PREFIX vivo: <http://vivoweb.org/ontology/core#>
       SELECT ?pt
       WHERE {
          ?individualUri <http://vivoweb.org/ontology/core#preferredTitle> ?pt
       }
       LIMIT 1
       """ .
```

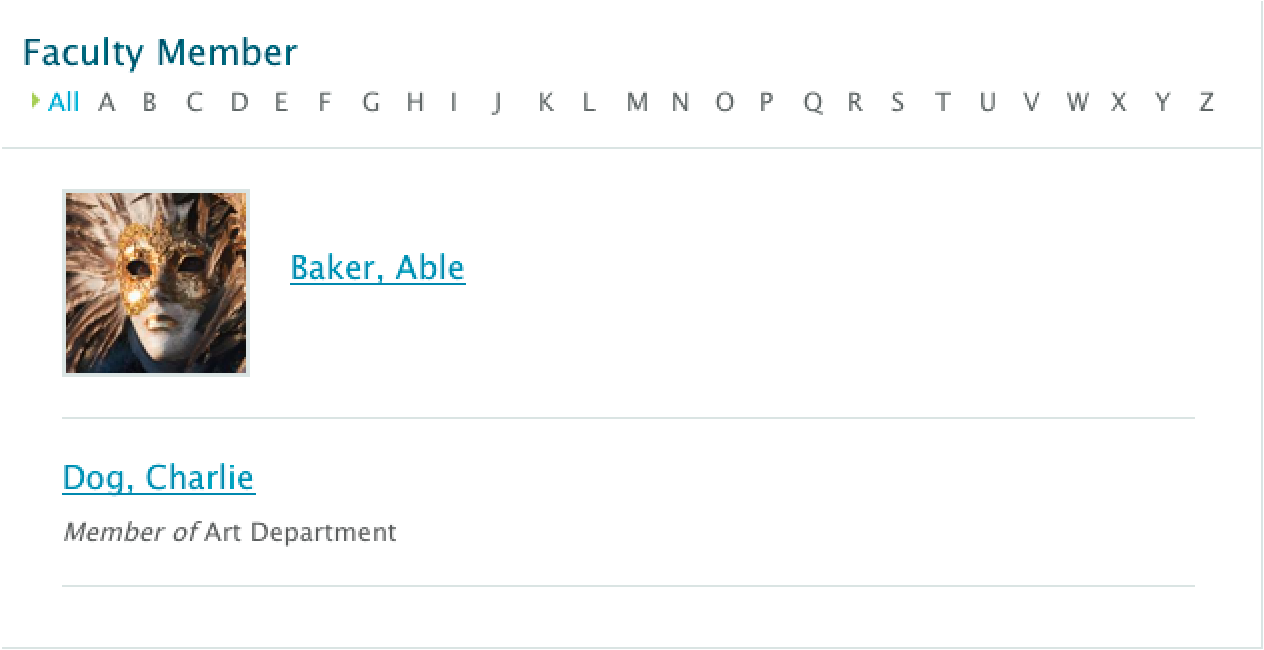
Note that the first DataGetter is the same as in the previous examples. If two custom short views want to use the same DataGetter, there is no need to code it twice. If two or more of these examples are tried at the same time, the short views would each refer to the same DataGetter.

Also note that this short view uses two DataGetters. One stores its results in "details" and the other stores its results in "extra", so the freemarker template will have access to both sets of results.

Create the template `view-browse-faculty.ftl` to look like this:

```
<#import "lib-properties.ftl" as p>

<li class="individual" role="listitem" role="navigation">

<#if (individual.thumbUrl)??>
    <img src="${individual.thumbUrl}" width="90" alt="${individual.name}" />
    <h1 class="thumb">
        <a href="${individual.profileUrl}" title="View the profile page for
               ${individual.name}}">${individual.name}</a>
    </h1>
<#else>
    <h1>
        <a href="${individual.profileUrl}" title="View the profile page for
               ${individual.name}}">${individual.name}</a>
    </h1>
</#if>

<#if (extra[0].pt)?? >
    <span class="title">${extra[0].pt}</span>
<#else>
    <#assign cleanTypes =
       'edu.cornell.mannlib.vitro.webapp.web.TemplateUtils$DropFromSequence'?new()
(individual.mostSpecificTypes, vclass) />
    <#if cleanTypes?size == 1>
        <span class="title">${cleanTypes[0]}</span>
    <#elseif (cleanTypes?size > 1) >
        <span class="title">
            <ul>
                <#list cleanTypes as type>
                <li>${type}</li>
                </#list>
            </ul>
        </span>
    </#if>
</#if>

<#if (details[0].deptName)?? >
    <span class="title"><em>Member of</em> ${details[0].deptName}</span>
</#if>

</li>
```

The new browse results look like this:



Faculty Member

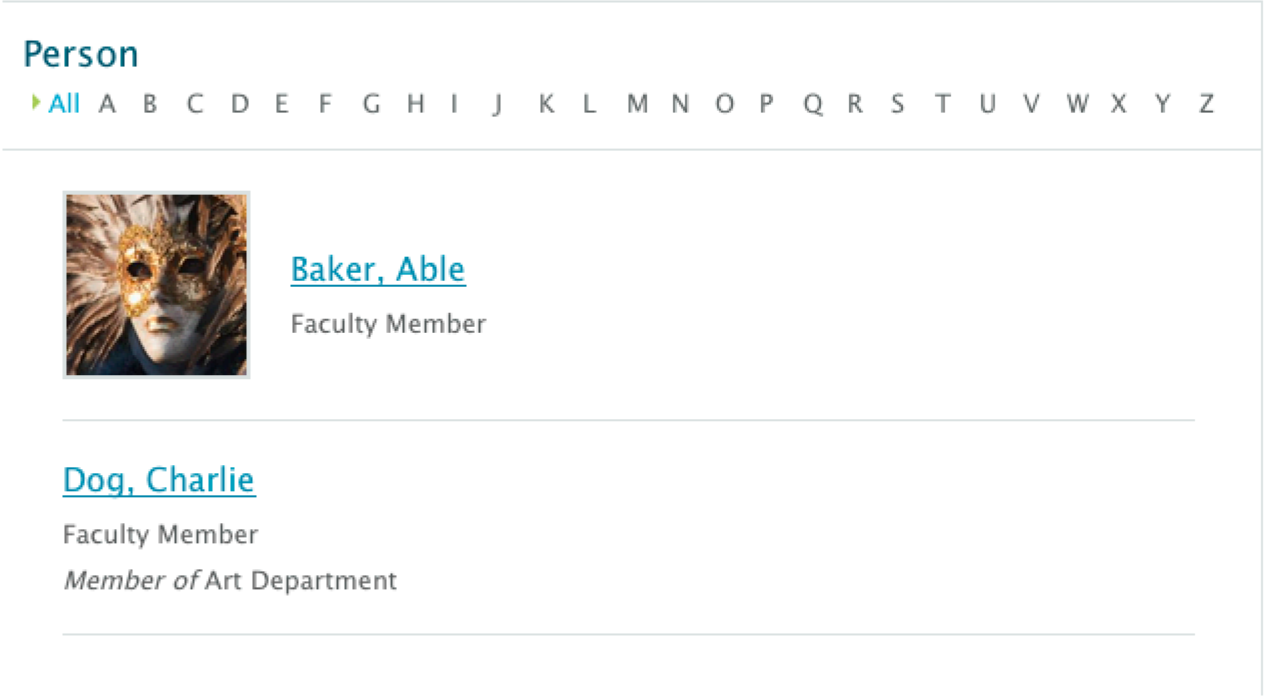
All A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Baker, Able

Dog, Charlie

Member of Art Department

Or this:



Person

All A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Baker, Able

Faculty Member

Dog, Charlie

Faculty Member

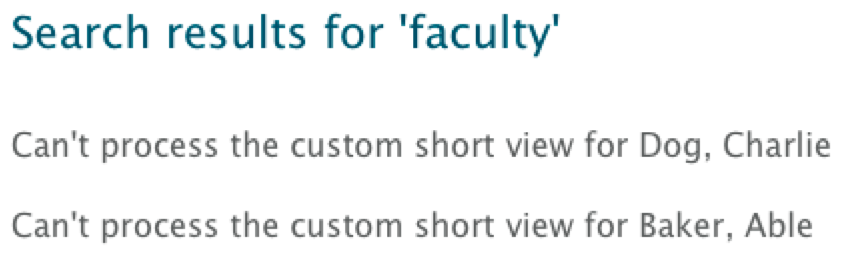
Member of Art Department

8.3.10.4 Troubleshooting

8.3.10.4.1 Errors in the template?

If the freemarker template for a short view contains syntax errors, the request will not throw an exception, which will be written to the VIVO log (`vivo.all.log`). The page will still render, but with an error message in place of the intended short view.

For example, in search results:

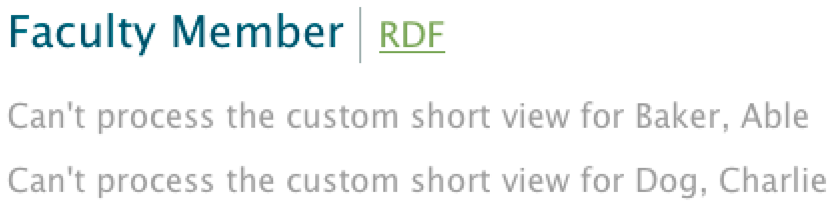


Search results for 'faculty'

Can't process the custom short view for Dog, Charlie

Can't process the custom short view for Baker, Able

In an index page:

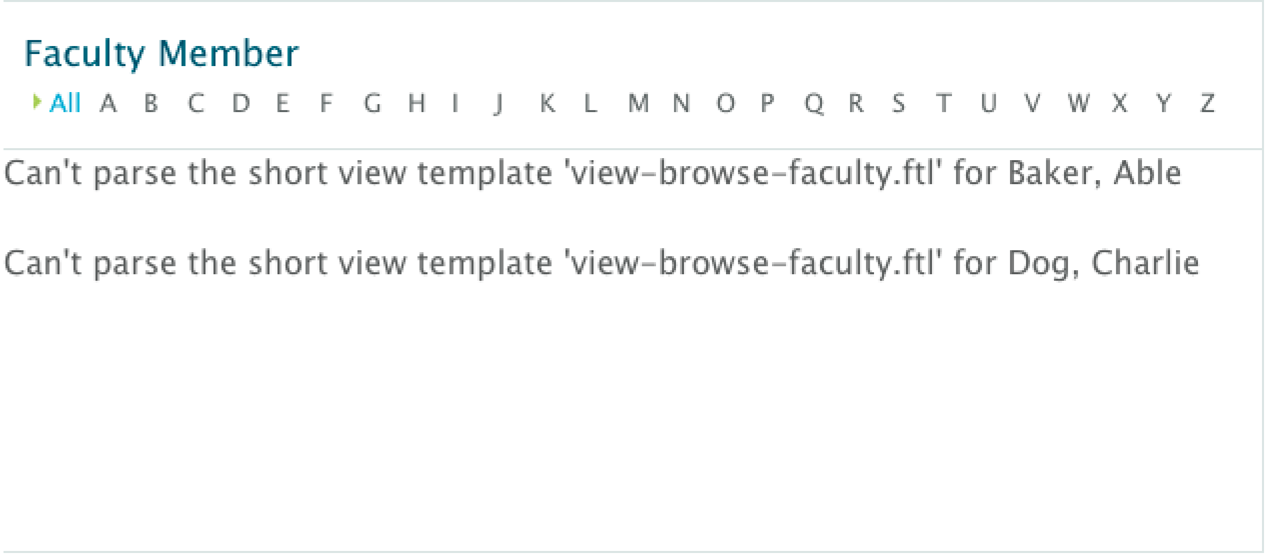


Faculty Member | RDF

Can't process the custom short view for Baker, Able

Can't process the custom short view for Dog, Charlie

In browse results:



Faculty Member

Can't parse the short view template 'view-browse-faculty.ftl' for Baker, Able

Can't parse the short view template 'view-browse-faculty.ftl' for Dog, Charlie

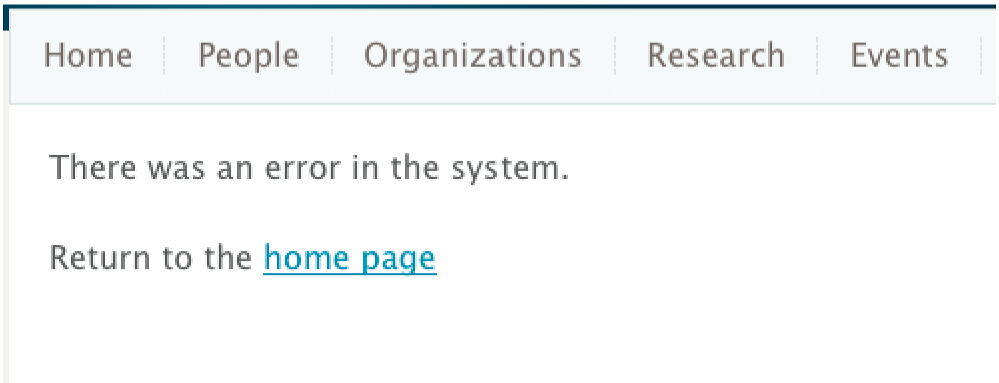
8.3.10.4.2 Errors in the Query?

If the SPARQL query defined in the configuration file contains syntax errors, the log will contain a stack trace for the exception. The information in the exception may be cryptic, but will at least tell you where the error is located within the query.

For example:

```
2012-10-23 17:25:26,499 ERROR [FreemarkerHttpServlet]
com.hp.hpl.jena.query.QueryParseException:
    Encountered " <VAR1> "?individualUri "" at line 6, column 1.
    Was expecting:
    "{" ...
```

The page will not render properly. Instead it will show a standard error screen:



There was an error in the system.

Return to the home page

8.3.10.4.3 Errors in the config file?

Other errors in the configuration file may give less obvious results. For example, if your customView object calls for a data getter that does not exist, the page will attempt to render without that data. If the data from that data getter is optional, you will see no error indicator except for a message in `vivo.all.log`

8.3.10.5 Notes

8.3.10.5.1 Waiting for the Application and Display Ontology

This implementation of short views is intended to be temporary, pending the implementation of the Application and Display Ontology (A&DO).

Much of the RDF that is entered in the configuration file (`shortview_config.n3`) should be replaced by triples in the A&DO. It's not clear where the SPARQL queries will be specified.

8.3.10.5.2 Classes are not inferred

Short views are applied based on the most-specific classes of the Individual. No inference is done when trying to find applicable views. So if an Individual has a type of FacultyMember, then a short view that applies to Person will not be used. Even though the Indvidual should also have a type of Person, it will not be among the most specific types for the Individual, and so does not apply. If you want a short view to apply to all sub-classes of Person, you must explicitly list each of these sub-classes in `shortview_config.n3`

This is expected to change when the Application and Display Ontology is used.

8.3.10.5.3 More than one applicable short view

In theory, it is possible that an Individual may qualify for two short views simultaneously. An Individual could have two most specific types (say FacultyMember and ExemptEmployee), and both of those types might have configured short views. Or, in the degenerate case, there might be multiple short views configured for a single type.

In these cases, one of the applicable short views will be arbitrarily selected.

8.3.10.5.4 Hard-coded BROWSE view for People

In the `BROWSE` context, if no short view is found for a given class URI, but that class is included in the People classgroup, a hard-coded short view is applied. This is to maintain compatibility with previous versions.

It is hoped that the Application and Display Ontology will be expressive enough to configure this behavior within the standard mechanism. Until then, it is coded into the class `edu.cornell.mannlib.vitro.webapp.services.shortview.FakeApplicationOntologyService`

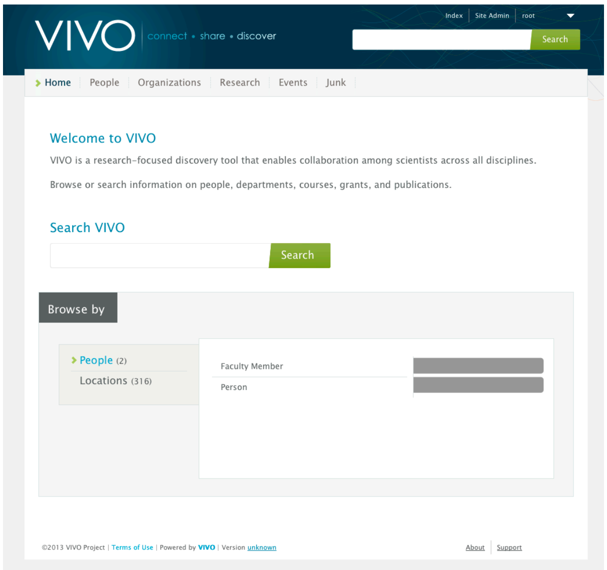
8.3.11 Creating a custom theme

8.3.11.1 Overview

8.3.11.1.1 What can it do for you?

Change the "look and feel" of your VIVO installation. Change the styling, the images, the layout, the text, and more. Modify the header and footer on all pages.

8.3.11.1.2 Before and After


VIVO
connect • share • discover
Index | Site Admin | root
Search
Home | People | Organizations | Research | Events | Junk

Welcome to VIVO

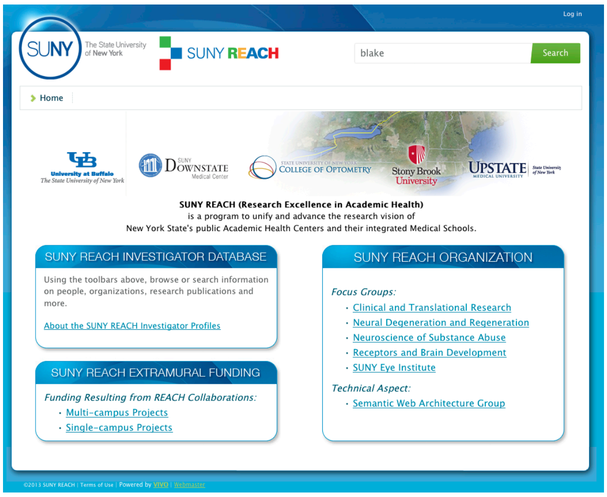
VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

Search
Browse by
People (2)
Locations (316)
Faculty Member
Person

©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown   About | Support


Log in
SUNY The State University of New York
SUNY REACH
blake
Search
Home
University at Buffalo The State University of New York
SUNY Downstate Medical Center
State University of New York College of Optometry
Stony Brook University
Upstate Medical University State University of New York

SUNY REACH (Research Excellence in Academic Health)
is a program to unify and advance the research vision of
New York State's public Academic Health Centers and their integrated Medical Schools.

SUNY REACH INVESTIGATOR DATABASE

Using the toolbars above, browse or search information on people, organizations, research publications and more.

About the SUNY REACH Investigator Profiles

SUNY REACH EXTRAMURAL FUNDING

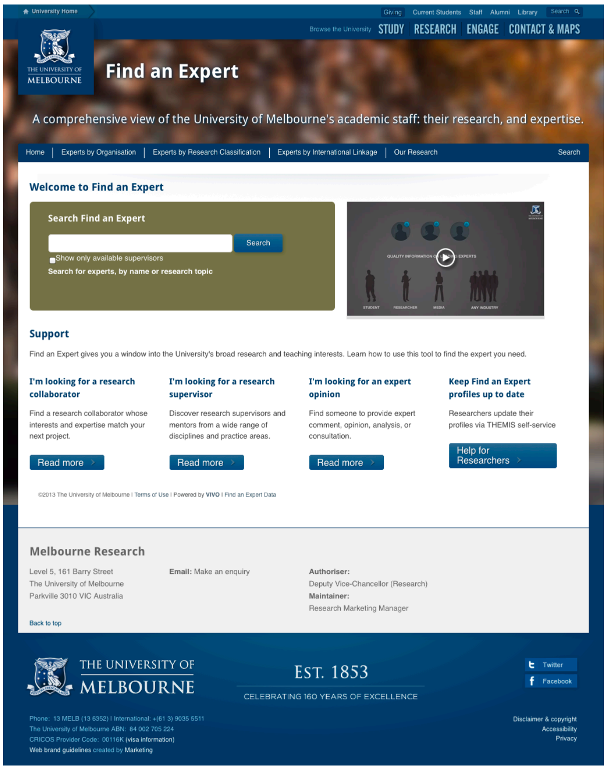
Funding Resulting from REACH Collaborations:
- Multi-campus Projects
- Single-campus Projects

SUNY REACH ORGANIZATION

Focus Groups:
- Clinical and Translational Research
- Neural Degeneration and Regeneration
- Neuroscience of Substance Abuse
- Receptors and Brain Development
- SUNY Eye Institute

Technical Aspect:
- Semantic Web Architecture Group

©2013 SUNY REACH | Terms of Use | Powered by VIVO | Webmaster


University Home
Giving | Current Students | Staff | Alumni | Library | Search
Browse the University STUDY RESEARCH ENGAGE CONTACT & MAPS
The University of Melbourne

Find an Expert

A comprehensive view of the University of Melbourne's academic staff: their research, and expertise.

Home | Experts by Organisation | Experts by Research Classification | Experts by International Linkage | Our Research | Search

Welcome to Find an Expert

Search Find an Expert

Search
☐ Show only available supervisors
QUALITY INFORMATION ON LEADING EXPERTS
STUDENT
RESEARCHER
MEDIA
ANY INDUSTRY

Search for experts, by name or research topic

Support

Find an Expert gives you a window into the University's broad research and teaching interests. Learn how to use this tool to find the expert you need.

I'm looking for a research collaborator

Find a research collaborator whose interests and expertise match your next project.

Read more

I'm looking for a research supervisor

Discover research supervisors and mentors from a wide range of disciplines and practice areas.

Read more

I'm looking for an expert opinion

Find someone to provide expert comment, opinion, analysis, or consultation.

Read more

Keep Find an Expert profiles up to date

Researchers update their profiles via THEMIS self-service

Help for Researchers

©2013 The University of Melbourne I Terms of Use I Powered by VIVO I Find an Expert Data

Melbourne Research

Level 5, 161 Barry Street
The University of Melbourne
Parkville 3010 VIC Australia

Email: Make an enquiry

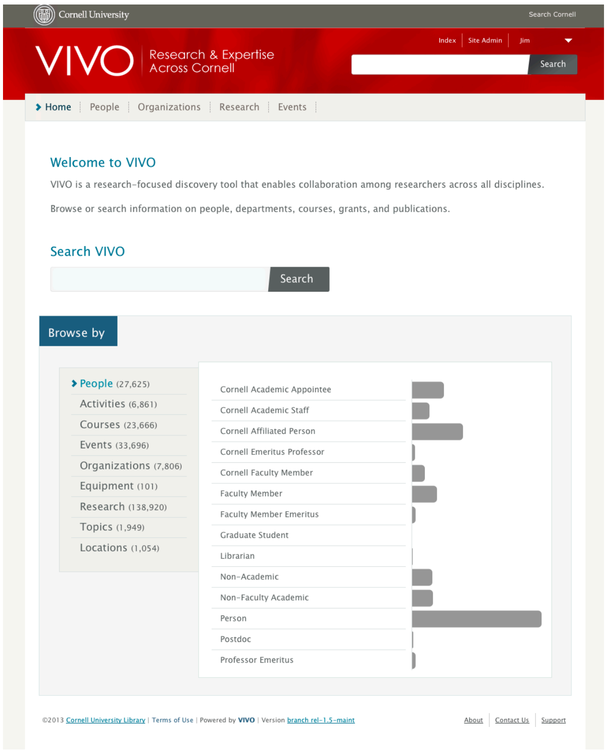
Authoriser:
Deputy Vice-Chancellor (Research)
Maintainer:
Research Marketing Manager

Back to top

The University of Melbourne
Est. 1853
Celebrating 160 years of excellence
Twitter
Facebook

Phone: 13 MELB (13 6352) I International: +(61 3) 9035 5511
The University of Melbourne ABN: 84 002 705 224
CRICOS Provider Code: 00116K (visa information)
Web brand guidelines created by Marketing

Disclaimer & copyright
Accessibility
Privacy


Cornell University
Search Cornell
Index | Site Admin | Jim
VIVO Research & Expertise Across Cornell
Search
Home | People | Organizations | Research | Events

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among researchers across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

Search
Browse by
People (27,625)
Activities (6,861)
Courses (23,666)
Events (33,696)
Organizations (7,806)
Equipment (101)
Research (138,920)
Topics (1,949)
Locations (1,054)
Cornell Academic Appointee
Cornell Academic Staff
Cornell Affiliated Person
Cornell Emeritus Professor
Cornell Faculty Member
Faculty Member
Faculty Member Emeritus
Graduate Student
Librarian
Non-Academic
Non-Faculty Academic
Person
Postdoc
Professor Emeritus

©2013 Cornell University Library | Terms of Use | Powered by VIVO | Version branch rel-1.5-maint   About | Contact Us | Support

8.3.11.1.3 What do you need to know?

- Standard web-site technologies: HTML, CSS and maybe JavaScript.

- Something about the Freemarker[114] template engine.
- Where the theme files are stored in VIVO, and how to reference them.
- Basic knowledge of semantic web technologies if some new user interface labels should be defined/changed

8.3.11.1.4 Getting started

VIVO comes with a standard theme, called `wilma.` `wilma` is in the folder in `vivo/installer/webapp/target/vivo/themes`.

To create a new theme, choose a name for your new theme. In these examples below we will call the new theme `fred`.

Copy the `wilma` directory and its contents to a new directory called `fred`. `fred` must also be in `vivo/installer/webapp/target/vivo/themes.`

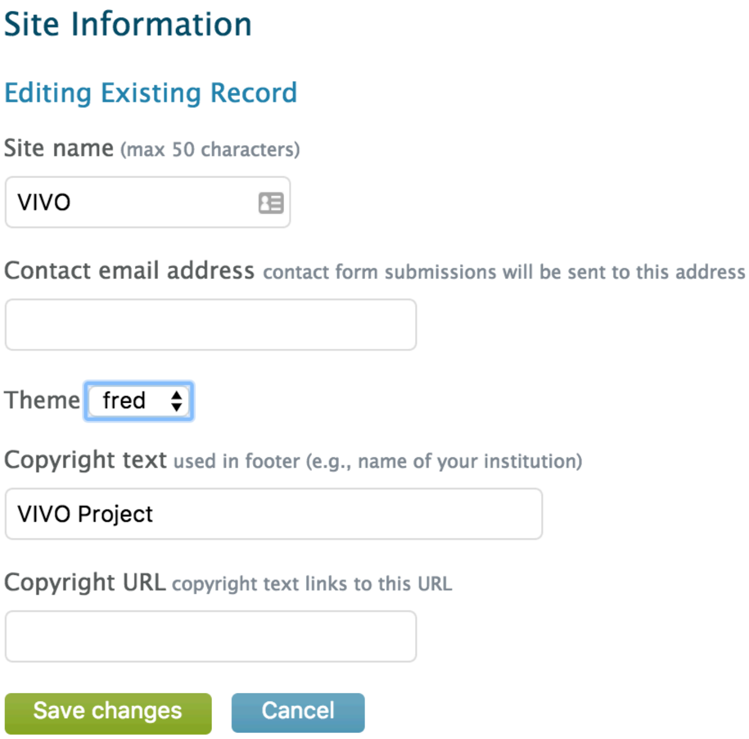
Your new theme will contain CSS files, image files, and Freemarker[115] templates.

Copy the `vivo/installer/home/target/vivo/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_wilma.ttl into vivo/installer/home/target/vivo/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_fred.ttl`, and substitute (find and replace) the term `wilma` with the term `fred` inside the new file. The previous example is working for English (en_US) user interface, the same action should be done for other languages used in the VIVO instance.

Run the Maven install to deploy your new theme to the Tomcat container. Restart the VIVO Tomcat process. You can then go to the ***Site Admin*** page and choose ***Site Information***, to select your theme as the current one.

---

[114] http://freemarker.org/
[115] http://freemarker.org/



Site Information

Editing Existing Record

Site name (max 50 characters): VIVO

Contact email address contact form submissions will be sent to this address:

Theme: fred

Copyright text used in footer (e.g., name of your institution): VIVO Project

Copyright URL copyright text links to this URL:

8.3.11.2 The structure of pages in VIVO

The pages in VIVO are built around three different frameworks. Each of these uses the same header and footer, to provide consistency. In addition to including the header and footer, the pages frequently include smaller templates to provide detail.

These are the basic frameworks:

The home page	As the point of entry for VIVO, the home page is special. It is based on the Freemarker template `page-home.ftl`
All other public pages	Based on the Freemarker template `page.ftl`
"back-end" pages	Pages used for editing the ontology, or manipulating the raw data of VIVO are based on a JSP named `basicPage.jsp`

8.3.11.3 Some significant templates

page.ftl

`page.ftl` is the default base template. The rest of the theme templates listed are components of `page.ftl` (included either directly or indirectly). Closer inspection of `page.ftl` reveals a stripped down file that declares minimal markup itself and instead reads as a list of includes for the component templates.

On the VIVO home page, page-home.ftl is used instead of page.ftl. It serves much the same purpose, but allows you to create a different layout for your home page than for the other pages in VIVO.

For consistency, It is critical that the following components be maintained:

head.ftl

This component template is responsible for everything within the `<head>` element. Note that the open and closing tags for the `<head>` element are defined in `page.ftl` and wrap the include for `head.ftl`. There are several includes within `head.ftl` that should be carried over to any new theme to maintain expected functionality:

- `<#include "stylesheets.ftl">` - ensures that the necessary stylesheets called by templates downstream will be added to the page via <link> elements
- `<#include "headscripts.ftl">` - ensures that the scripts called by templates which must be in the <head> will be added to the page via <script> elements

identity.ftl

This component template is responsible for rendering the VIVO logo, secondary navigation and search input field at the top of the page. There are no mandatory includes from `identity.ftl` that need to be carried over but there are 2 template variables that are of particular interest (`${user}` and `${urls}`).

menu.ftl

This component template is responsible for rendering the primary navigational menu for the site. In wilma, it also happens to declare the open tag for the main content container. There are no mandatory includes from `menu.ftl`. The `${menu`} template variable is crucial since it contains an array of menu items needed to build the primary navigational menu.

footer.ftl

This component template is responsible for rendering the copyright notice, revision information, secondary navigation, and link for the contact form. There is a single include that should be maintained:

- `<#include "scripts.ftl">` - ensures that the non head scripts (those that don't need to be placed in the `<head>`) called by the templates will be added to the page via `<script>` elements

Several template variables of interest include `${copyright}`, `${user}`, and `${version}`.

googleAnalytics.ftl

This component template is included by `footer.ftl`. Simply uncomment the `<script>` element and provide your Google Analytics Tracking Code.

Adjust the markup as necessary in `page.ftl,` and these component templates to achieve the desired content structure, and modify the stylesheets to meet layout needs and style your site. Remember that changes should be made in the source directory and that you will need to redeploy the project before the changes are reflected in the live website.

You can find more information about the structure of the VIVO theme in How VIVO creates a page (see page 248).

8.3.11.4 Making changes

8.3.11.4.1 Modify files in the theme

You can edit the Freemarker templates and the CSS files in the theme with any text editor. You can replace the image files with images that you choose.

8.3.11.4.2 Add files to the theme

8.3.11.4.2.1 Add CSS, JavaScript, or image files

As you modify the templates, you may want to use additional images, CSS files, or JavaScript files. When your templates refer to these files, they will use the Freemarker variable `urls.theme`, as shown in these examples:

```
<!-- an image file -->
<img src="${urls.theme}/images/arrow-green.gif"/>

<!-- a CSS file -->
<link rel="stylesheet" href="${urls.theme}/css/screen.css" />

<!-- a JavaScript file (create a js directory in your theme) -->
<script type="text/javascript" src="${urls.theme}/js/my.js"></script>
```

8.3.11.4.2.2 Add Freemarker templates

If your modifications use new Freemarker templates, you can refer to them more simply. Freemarker already knows where your theme directory is located.

```
<#include "my-new-template.ftl">
```

8.3.11.4.3 Override files that are not in the theme directory

In order to keep the theme directory uncluttered, VIVO keeps most of the front-end files in a separate location. Changes to the theme usually involve the files in the theme directory, but you can override other files as well.

8.3.11.4.3.1 Override CSS, JavaScript or image files that are not in the theme directory

You may notice that templates refer to files that are not in the theme directory. They use references based on the Freemarker variable `urls.base` instead of `urls.theme`, like this:

```
<!-- an image file -->
<img src="${urls.base}/images/arrowIcon.gif"/>

<!-- a CSS file -->
<link rel="stylesheet" href="${urls.base}/css/login.css" />

<!-- a JavaScript file -->
<script type="text/javascript" src="${urls.base}/js/browserUtils.js"></script>
```

These refer to files in the `vivo/installer/webapp/target/vivo` directory. If you look, you will see that `this directory` contains some files also used in the construction of thr VIVO interface.

8.3.11.4.3.2 Override Freemarker templates that are not in the theme directory

To override templates not in the theme directory, simply modify freemarker templates in `vivo/installer/webapp/target/vivo.` These changes will apply to all your themes.

VIVO treats all available Freemarker templates as belonging to the same flat namespace, whether they are in the theme directory or in the `templates/freemarker` directory, or one of its sub-directories. A file in `vivo/installer/webapp/target/vivo` can be overridden by a corresponding file in the theme directory.

8.3.11.4.4 Working on the theme

When you make changes to VIVO, you should make the changes in your VIVO distribution directory, run Maven install, restart Tomcat, and test the changes. If you are doing full customizing of VIVO, this cycle might be best.

If you are only working on the theme, you can speed things up.

- Tell the build script to skip the unit tests: they don't test the theme
  - `mvn install -Dskiptests=true`
- Don't restart Tomcat
  - VIVO always serves the most recent version of CSS files, image files, and JavaScript files. You don't need to restart Tomcat to make that happen.
  - However, your browser may cache these files so you won't see the most recent version. Here are some suggestions for bypassing your browser cache[116].

---

[116] http://en.wikipedia.org/wiki/Wikipedia:Bypass_your_cache

- Tell VIVO to reload Freemarker templates each time they are requested. See Tips for Interface Developers (see page 259).

Some developers prefer to make theme changes inside the `tomcat/webapp/vivo` directory. This eliminates the need to run the build script, but opens the threat of having the changes over-written the next time the build script runs.

8.3.11.4.4.1 When to restart Tomcat

If you make changes to any of the source files in the theme, including images, CSS, JavaScript or Freemarker templates, you must run the build script, but you do not need to restart Tomcat.

8.3.12 Creating custom entry forms

8.3.12.1 Overview

Custom entry forms allow VIVO to transcend the general-purpose, utilitarian editing scheme of Vitro. Without custom entry forms, VIVO users must edit each RDF triple individually. With a custom entry form, users can edit a complex data structure on a single page.

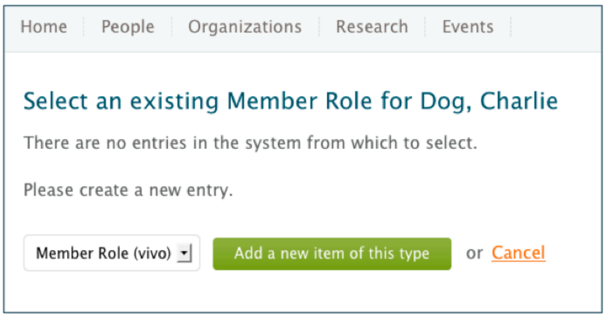
VIVO is distributed with a dozens of custom entry form generators. You may want to modify these form generators, or add more of your own.

8.3.12.2 An example

Say you wish to establish that a particular person is a member of a particular academic department. This relationship can be expressed as a member role. See Membership Model (see page 474)

But what if the academic department doesn't exist in VIVO yet? You will want to create that department, and assign a name to it. You may also want to record the member role in that department, when their membership began, and when it ended (if it is not ongoing).

Without a custom entry form, you would need to record each piece of data individually.

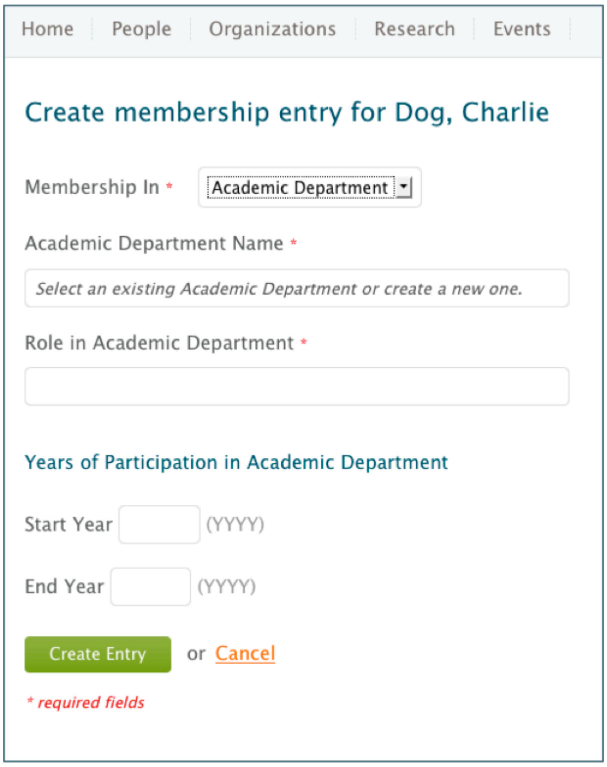


Select an existing Member Role for Dog, Charlie

There are no entries in the system from which to select.

Please create a new entry.

VIVO includes a custom form generator for this relationship. The custom entry form looks like this:



Create membership entry for Dog, Charlie

Membership In * Academic Department

Academic Department Name *

Select an existing Academic Department or create a new one.

Role in Academic Department *

Years of Participation in Academic Department

Start Year (YYYY)

End Year (YYYY)

** required fields*

8.3.12.3 How is it created?

The creation of custom entry forms is an arcane and eldritch art, for which little documentation is available.

Each form requires a Java class known as a `EditConfigurationGenerator`. The generator describes the data structure being created, lists the SPARQL queries used, and includes a reference to the Freemarker template that will render the form.

You can start by examining the existing generators in this directory `[VIVO]/api/src/main/java/edu/cornell/mannlib/vitro/webapp/edit/n3editing/configuration/generators`

and the Freemarker templates found here `[VIVO]/webapp/src/main/webapp/templates/freemarker/edit/forms`

There is also a short page of technical description called Implementing custom forms using N3 editing (see page 228).

—

8.3.12.4 Accessing VIVO Data Models

8.3.12.4.1 Accessing the models

There is an incredible variety of ways to access all of these models. Some of this variety is because the models are accessed in different ways for different purposes. Additional variety stems from the evolution of VIVO in which new mechanisms were introduced without taking the time and effort to phase out older mechanisms.

Here are some of the ways for accessing data models:

8.3.12.4.1.1 Attributes on Context, Session, or Request

Previously, it was common to assign a model to the ServletContext, to the HTTP Session, or to the HttpSessionRequest like this:

```
OntModel ontModel = (OntModel) getServletContext().getAttribute("jenaOntModel");

Object sessionOntModel = request.getSession().getAttribute("jenaOntModel");
```

Occasionally, conditional code was inserted, to retrieve a model from the Request if available, and to fall back to the Session or the Context as necessary. Such code was sporadic, and inconsistent. This sort of model juggling also involved inversions of logic, with some code acting so a model in the Request would override one in the Session, while other code would prioritize the Session model over the one in the Request. For example:

```
public OntModel getDisplayModel(){
    if( _req.getAttribute("displayOntModel") != null ){
        return (OntModel) _req.getAttribute(DISPLAY_ONT_MODEL);
    } else {
        HttpSession session = _req.getSession(false);
        if( session != null ){
            if( session.getAttribute(DISPLAY_ONT_MODEL) != null ){
                return (OntModel) session.getAttribute(DISPLAY_ONT_MODEL);
            }else{
                if( session.getServletContext().getAttribute(DISPLAY_ONT_MODEL) !=
null){
                    return
 (OntModel)session.getServletContext().getAttribute(DISPLAY_ONT_MODEL);
                }
            }
        }
    }
    log.error("No display model could be found.");
    return null;
}
```

This mechanism has been removed in 1.6, being subsumed into the `ModelAccess` class (see below). Now, the `ModelAccess` attributes on Request, Session and Context are managed using code that is private to `ModelAccess` itself. Similarly, the code which gives priority to a Request model over a Session model is uniformly implemented across the models.

It remains to be seen whether this uniformity can satisfy the various needs of the application. If not, at least the changes can all be made within a single point of access.

8.3.12.4.1.2 The DAO layer

This mechanism is pervasive through the code, and remains quite useful. In it, a `WebappDaoFactory` is created, with access to particular data models. This factory then can be used to create DAO objects which satisfy interfaces like `IndividualDao`, `OntologyDAO`, or `UserAccountsDAO`. Each of these object implements a collection of convenience methods which are used to manipulate the backing data models.

Because the factory and each of the DAOs is an interface, alternative implementations can be written which provide

- Optimization for Jena RDB models
- Optimization for Jena SDB models
- Filtering of restricted data
- and more...

Initially, the `WebappDaoFactory` may have been used only with the full Union model. But what if you want to use these DAOs only against asserted triples? Or only against the ABox? This led to the `OntModelSelector`.

8.3.12.4.1.3 OntModelSelectors

An `OntModelSelector` provides a way to collect a group of Models and construct a `WebappDaoFactory`. With slots for ABox, TBox, and Full model, an `OntModelSelector` could provide a consistent view on assertions, or on inferences, or on the union. The `OntModelSelector` also holds references to a display model, an application metadata model, and a user accounts model, but these are more for convenience than flexibility.

Prior to release 1.6, `OntModelSelectors`, like `OntModel` s, were stored in attributes of the Context, Session, and Request. They have been subsumed into the `ModelAccess` class.

Further, the semantics of the "standard" `OntModelSelectors` have changed, so they only act as facades before the Models store in `ModelAccess`. In this way, if we make this call:

```
ModelAccess.on(session).setOntModel(ModelID.BASE_ABOX, someWeirdModel)
```

Then both of the following calls would return the same model:

```
ModelAccess.on(session).getOntModel(ModelID.BASE_ABOX);
ModelAccess.on(session).getBaseOntModelSelector().getABoxModel();
```

Again, this is a change in the semantics of OntModelSelectors. It insures a consistent representation of `OntModel`s across `OntModelSelector`s, but it is certainly possible that existing code relies on an inconsistent model instead.

8.3.12.4.1.4 The RDF Service

8.3.12.4.1.5 Model makers and Model sources

8.3.12.4.2 The ModelAccess class

TBD - Show how it represents all of these distinctions. Describe the scope searching and masking, wrt set and get. Include the OntModelSelectors and WADFs.

8.3.12.4.3 Initializing the Models

When VIVO starts up, `OntModel` objects are created to represent the various data models. The configuration models are created from the datasource connection, usually to a MySQL database. The content models are created using the new RDFService layer. By default this also uses the datasource connection, but it can be configured to use any SPARQL endpoint for its data.

Some of the smaller models are "memory-mapped" for faster access. This means that they are loaded entirely into memory at startup. Any changes made to the memory image will be replicated in the original model.

The data in each model persists in the application datasource (usually a MySQL database), or in the RDFService. Also, data from disk files may be loaded into the models. This may occur:

- the first time that VIVO starts up,
- if a model is found to be empty,
- every time that VIVO starts up.

depending on the particular model.

8.3.12.4.3.1 Where are the RDF files?

In the distribution, the RDF files appear in `[vivo]/rdf` and in `[vitro]/webapp/rdf`. These directories are merged during the build process in the usual way, with files in VIVO preferred over files in Vitro.

During the build process, the RDF files are copied to the VIVO home directory, and at runtime VIVO will read them from there.

8.3.12.4.3.2 The "first time"

For purposes of initialization, "first time" RDF files are loaded if the relevant data model contains no statements. Content models may also load "first time" files if the RDFService detects that its SDB-based datastore has not been initialized.

8.3.12.4.3.3 Initializing Configuration models

Application metadata

Function: Describes the configuration of VIVO at this site. Many of the configuration options are obsolete.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this is the first startup, read the files in rdf/applicationMetadata/firsttime.

- In Vitro, there are none
- In VIVO, initialSiteConfig.rdf, classgroups.rdf and propertygroups.rdf

User Accounts

Contains login credentials and assigned roles for VIVO users.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-userAccounts

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/auth/firsttime.

- In Vitro, there are none (except during Selenium testing)
- In VIVO, there are none.

Every time, read the files in rdf/auth/everytime

- In Vitro, permissions_config.n3
- In VIVO, there are none.

The Display model

This is the ABox for the display model, and contains the RDF statements that define managed pages, custom short views, and other items.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/display/firsttime

- In Vitro, application.owl, menu.n3, profilePageType.n3
- VIVO contains its own copy of menu.n3, which overrides the one in Vitro

Every time, read the files in rdf/display/everytime

- in Vitro, displayModelListViews.rdf
- In VIVO, homePageDataGetters.n3, localeSelectionGUI.n3, vivoDepartmentQueries.n3, vivoListViewConfig.rdf, vivoSearchProhibited.n3

Display TBox

The TBox for the display model.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadataTBOX

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayTbox/everytime.

- In Vitro, displayTBOX.n3
- In VIVO, there are none

DisplayDisplay

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata-displayModel

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayDisplay/everytime

- In Vitro, displayDisplay.n3
- In VIVO, there are none.

8.3.12.4.3.4 Initializing Content models

base ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-2

Source: named graph from the RDFService

If first setup, read the files in rdf/abox/firsttime

- In Vitro, there are none
- In VIVO, geopolitical.ver1.1-11-18-11.individual-labels.rdf

Every restart, read the files in rdf/abox/filegraph, and create named models in the RDFService. Add them as sub-models to the base ABox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, there are none
- In Vivo, geopolitical.abox.ver1.1-11-18-11.owl, academicDegree.rdf, continents.n3 us-states.rdf, dateTimeValuePrecision.owl, validation.n3, documentStatus.owl, vocabularySource.n3

base TBox

Name: http://vitro.mannlib.cornell.edu/default/asserted-tbox

Source: named graph from the RDFService (memory-mapped)

If first setup, read the files in rdf/tbox/firsttime (without subdirectories)

- In Vitro, there are none
- In VIVO, additionalHiding.n3 initialTBoxAnnotations.n3

Every restart, read the files in rdf/tbox/filegraph, and create named models in the RDFService. Add them as sub-models to the base TBox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, vitro-0.7.owl, vitroPublic.owl
- In VIVO, 44 files:

/usr/local/vivo/home/rdf/tbox/filegraph

```
README.md                   education.owl           personTypes.n3
agent.owl                   event.owl               process.owl
appControls-temp.n3         geo-political.owl       publication.owl
bfo-bridge.owl              grant.owl               relationship.owl
bfo.owl                     linkSuppression.n3      relationshipAxioms.n3
classes-additional.owl      location.owl            research-resource-iao.owl
clinical.owl                object-properties.owl   research-resource.owl
contact-vcard.owl           object-properties2.owl  research.owl
contact.owl                 object-properties3.owl  role.owl
data-properties.owl         objectDomains.rdf       sameAs.n3
dataDomains.rdf             objectRanges.rdf        service.owl
dataset.owl                 ontologies.owl          skos-vivo.owl
date-time.owl               orcid-interface.n3      teaching.owl
dateTimeValuePrecision.owl  other.owl               vitro-0.7.owl
documentStatus.owl          outreach.owl            vitroPublic.owl
```

base Full

Source: a combination of base ABox and base TBox

inference ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-inf

Source: named graph from the RDFService

inference TBox

Name: http://vitro.mannlib.cornell.edu/default/inferred-tbox

Source: named graph from the RDFService (memory-mapped)

inference Full

Source: a combination of inference ABox and inference TBox

union ABox

Source: a combination of base ABox and inference ABox

union TBox

Source: a combination of base TBox and inference TBox

union Full

Source: a combination of union ABox and union TBox

8.3.12.4.4 Transition from previous methods

TBD - What are we transitioning from? Check out VIVO-82.

- Semantics have changed: saves code, but may alter some uses.
  - Always searches the stack
  - OMS are facades with no internal state
    - There is no way to set an OMS - set the models instead
    - Keeps consistent

	prior to ModelAccess	**using ModelAccess**
User Accounts Model	ctx.getAttribute("userAccountsOntModel")	ModelAccess.on(ctx).getUserAccountsModel()
	ctx.setAttribute("userAccountsOntModel", model)	ModelAccess.on(ctx).setUserAccountsModel(model)
DisplayModel	req.getAttribute("displayOntModel")	ModelAccess.on(req).getDisplayModel()
	session.getAttribute("displayOntModel")	ModelAccess.on(session).getDisplayModel()

	prior to ModelAccess	**using ModelAccess**
	ctx.getAttribute("displayOntModel") ModelContext.getDisplayModel(ctx)	ModelAccess.on(ctx).getDisplayModel()
	ctx.setAttribute("displayOntModel", model) ModelContext.setDisplayModel(model, ctx)	ModelAccess.on(ctx).getDisplayModel()
	req.setAttribute("displayOntModel", model)	ModelAccess.on(req).setDisplayModel(model)
"jenaOntModel"	ctx.getAttribute("jenaOntModel")	ModelAccess.on(ctx).getJenaOntModel()
	session.getAttribute("jenaOntModel")	ModelAccess.on(session).getJenaOntModel()
	req.getAttribute("jenaOntModel")	ModelAccess.on(req).getJenaOntModel()
	ctx.setAttribute("jenaOntModel", model)	ModelAccess.on(ctx).setOntModel(ModelID.UNION_FULL, model)
	req.setAttribute("jenaOntModel", model)	ModelAccess.on(req).setOntModel(ModelID.UNION_FULL, model) ModelAccess.on(req).setJenaOntModel(model)
"baseOntModel" "assertionsModel" Base Full Model	ModelContext.getBaseOntModel(ctx) ctx.getAttribute("baseOntModel") session.getAttribute("baseOntModel")	ModelAccess.on(ctx).getOntModel(ModelID.BASE_FULL) ModelAccess.on(ctx).getBaseOntModel()
	ModelContext.setBaseOntModel(model, ctx)	

	prior to ModelAccess	**using ModelAccess**
"inferenceModel" Inference Full Model	ctx.getAttribute("inferenceOntModel")	ModelAccess.on(ctx).getInferenceOntModel()

Notes:

- "jenaOntModel" is a previous term for the Union Full model. The convenience methods `getJenaOntModel()` and `setJenaOntModel(m)` support this use.
- "baseOntModel" and "assertionsModel" are both previous terms for the Base Full model. The convenience methods `getBaseOntModel()` and `setBaseOntModel(m)` support this use.

	prior to ModelAccess	**using ModelAccess**
ontModelSelector unionOntModelSelector	ModelContext.setOntModelSelector(model, ctx) ModelContext.getUnionOntModelSelector(ctx) ctx.getAttribute("ontModelSelector") ctx.getAttribute("unionOntModelSelector")	no mutator methods ModelAccess.on(ctx).getOntModelSelector() ModelAccess.on(ctx).getUnionOntModelSelector()
baseOntModelSelector	ctx.getAttribute("baseOntModelSelector")	ModelAccess.on(ctx).getBaseOntModelSelector()
inferenceOntModelSelector	ctx.getAttribute("inferenceOntModelSelector")	ModelAccess.on(ctx).getInferenceOntModelSelector()

- The default WebappDaoFactory is the one backed by the unionOntModelSelector. On the request level, this is also known as the "fullWebappDaoFactory". The convenience methods `getWebappDaoFactory()` and `setWebappDaoFactory(wdf)` support this use.
- "baseWebappDaoFactory" and "assertionsWebappDaoFactory" are both previous terms for the WebappDaoFactory backed by the baseOntModelSelector. The convenience methods `getBaseWebappDaoFactory()` and `setBaseWebappDaoFactory(wdf)` support this use.
- Nobody was using the "deductionsWebappDaoFactory", so we got rid of it.

8.3.12.5 Implementing custom forms using N3 editing

8.3.12.5.1 Overview

The Vitro/VIVO system comes with basic RDF editing capabilities to add object and datatype statements to individuals. Frequently, people deploying Vitro/VIVO desire a web form which allows editing of multiple properties and individuals on the same form. A contact information form would be an example of a feature that would be implemented with a custom form in Vitro/VIVO.

The creation of a custom forms in Vitro/VIVO is done in two parts. The first is an implementation of the java interface EditConfigurationGenerator and the second is a FreeMarker template for the presentation. The EditConfigurationGenerator creates a EditConfiguration that controls how the values from the form will be used in the editing of the RDF, server side validation, which template to use, and other aspects of the edit. The FreeMarker template controls the HTML and Javascript for the form.

The EditConfigurationGenerator classes can be associated with an RDF property so that they are used from an individual’s profile page or by a direct URL.

The main concept of custom forms is that the values submitted by the HTTP request will be substituted into placeholders in RDF N3 strings. These strings are then parsed to Jena RDF Model objects and that RDF is added to the system. For the modification of an existing value, a second set of strings is created and parsed which become the RDF statements to remove for the edit. This substitution is why the editing system is frequently called “N3 Editing”. In practice, the N3 strings use only the turtle subset of the N3 syntax.

8.3.12.5.2 Steps of an Edit

8.3.12.5.2.1 Step 1. Getting the link to the edit

When a user is logged in, individual profile pages have edit links next to the listed properties. These links will take the user to a page with an edit form. The links on the individual profile page are routed to the EditRequestDispatchController which will determine which EditConfigurationGenerator to use based on which property is being edited. The VIVO/Vitro system can be configured to associate a EditConfigurationGenerator with a property so that the edit links will use a custom EditConfigurationGenerator. If no custom form is specified then the default object or data property EditConfigurationGenerator will be used.

A property can be associated with a custom form in one of two ways:
A) if you go to the site admin -object property hierarchy - the property you want associated with the form, click on the property then edit property record, you can put in the Java class name of the generator in the custom entry form field. E.g.edu.cornell.mannlib.vitro.webapp.edit.n3editing.configuration.generators.AddDistributionGenerator. This will allow you to associate the custom form while the system is running.

B) if you will be deploying the system for the first time and starting with an empty database, you would update vivo-core-1.5-annotations.rdf to specify that the property has a custom form using the vitro:customEntryFormAnnot[117] property.

---

[117] http://vitrocustomEntryFormAnnot

8.3.12.5.2.2 Step 2. Generating the EditConfiguration

When the user clicks the link, the client browser requests the URL of the edit link which will be to the EditRequestDispatchControl. That servlet will set up all that is needed in the session for the edit and respond with the HTML form. A custom form is setup by the EditConfigurationGenerator which has the sole purpose of making an EditConfiguration object. The EditRequestDispatchController will run getEditConfiguration() on the EditConfigurationGenerator to create the EditConfiguration. The EditConfiguration object has properties to define the characteristics of the edit. The EditConfiguration will specify the FreeMarker template for the form, and the server side instructions for validating the submitted result and instructions for processing the edit. When authoring a custom form, a central task is the coding of the EditConfigurationGenerator to produce an EditConfiguration that encodes logic of how you desire the edit to happen. The EditConfigurationGenerator is just a java class that creates an EditConfiguration.

When generating the EditConfiguration at runtime, an edit key will be created and the completed EditConfiguration will be associated with that key in the server side user session. This edit key is used to handle parallel editing and back button complexity. The EditConfiguration object for an edit is in a one to one relation with the HTML form for an edit. If the user goes to edit a street address and then goes to edit that street address a second time, the first edit will have an EditConfiguration object in the session and an HTML form with one edit key, and the second will have a different EditConfiguration in the session and an HTML form with a different edit key. An HTML form created for a edit will have an “edit key“ to associate that specific instance of the HTML form with an object stored in the user’s session.

The EditConfiguration can specify SPARQL queries for existing values for fields of the form. These are executed as part of the generation of the EditConfiguration.

8.3.12.5.2.3 Step 3. HTML creation by FreeMarker

Once the EditRequestDispatchController has the EditConfiguration and put it in the session, it will set up some standard values for the template and pass them and the EditConfiguration to the FreeMarker template specified in EditConfiguration.getTemplate(). The HTML form is then generated using the normal FreeMarker process. The HTML form must contain a field with the EditKey so associate the edit with an EditConfiguration in the session.

8.3.12.5.2.4 Step 4. Response From Client

The form will be submitted by the client’s browser to ProcessRdfFormController. This will get the EditConfiguration based on the edit key from the submitted values. It will run validation and then substitute the values from the form into the N3 templates and parse the N3 to RDF. The N3 that gets created will be then added to the VIVO/Vitro models. If the edit is a change of an existing value, then the RDF for the statements to remove will be created and removed form the VIVO/Vitro models.

8.3.12.6 Servlet Lifecycle Management

Description

Like most Java Enterprise applications, Vitro servlets rely on the ServletContext to hold object that they will need to use when servicing requests. These objects are created by ServletContextListeners, which are run by the StartupManager.

The StartupManager creates instances of the listeners and runs them, accumulating information about their running in the StartupStatus.

As each listener runs, it may add messages to the StartupStatus. Each message will have a severity level associated with it:

- FATAL – The listener encountered a problem. Perhaps the application was configured incorrectly, or perhaps the system utilities are not performing as intended. The problem is severe enough that the application will not run. The message describes the problem, with suggestions on how to fix it.
- WARNING – The listener encountered a problem, but the problem will not prevent the application from running. The message describes the problem, and tells what parts of the application will be affected, with suggestions on how to fix the problem.
- INFO – No problem is indicated. The message contains information that may be helpful in monitoring the application.

If a FATAL status is recorded, the StartupManager will not execute any additional listeners. Access to the application will be blocked, and any attempt to access the application will display the StartupStatus in an error page.

If a WARNING status is recorded, the StartupManager continues as normal. Access to the application will be blocked one time, to display the StartupStatus. In subsequent requests, the application will respond normally.

When logged in, an administrator may view the StartupStatus from a link on the Site Admin page.

8.3.12.6.1 Specifying context listeners

In any Java Enterprise application, developers can specify context listeners in the deployment descriptor (web.xml). These listeners that will be activated when the application starts and when it shuts down.

In Vitro, the only listener in web.xml is the StartupManager. Here is the relevant section of Vitro’s web.xml:

```
<!--
   StartupManager instantiates and runs the listeners from startup_listeners.txt
   All ServletContextListeners should be listed there, not here.
-->
<listener>
  <listener-class>edu.cornell.mannlib.vitro.webapp.startup.StartupManager</listener-class>
</listener>
```

Vitro contains a list of startup listeners in a file at Vitro (see page 398)/webapp/config/startup_listeners.txt. This file is simple text with each line containing the fully-qualified class name of a startup listener. Blank lines are ignored, as are comment lines – lines that begin with a “hash” character. Here is a portion of that file:

```
#
# ServletContextListeners for Vitro,
# to be instantiated and run by the StartupManager.
#
```

```
edu.cornell.mannlib.vitro.webapp.config.ConfigurationPropertiesSetup

edu.cornell.mannlib.vitro.webapp.config.RevisionInfoSetup

edu.cornell.mannlib.vitro.webapp.email.FreemarkerEmailFactory$Setup

# DefaultThemeSetup needs to run before the JenaDataSourceSetup to allow creation
# of default portal and tab
edu.cornell.mannlib.vitro.webapp.servlet.setup.DefaultThemeSetup
```

8.3.12.6.2 Writing context listeners

Each listener must implement the ServletContextListener interface, and must have a zero-argument constructor.

When Vitro starts, the StartupManager will call contextInitialized() in each listener, in the order that they appear in the file. The listener can call methods on StartupStatus to record messages. If the listener is successful, it should record one or more INFO messages that provide a brief description of what it has done. If a problem is detected, the listener may record WARNING messages or ERROR messages, depending on the severity of the problem. The listener may also throw a RuntimeException from contextInitialized(), which the StartupManager will treat like an ERROR.

Here is an example of a basic listener. When contextInitialized() is called, the listener will perform some setup. If there is no problem, a call to StartupStatus.info() reports some basic information about the listener's actions. If a problem is found, a call to StartupStatus.warning() describes the nature of the problem (by reporting the exception) and how this problem will affect the application.

```
public static class Setup implements ServletContextListener {
   @Override
   public void contextInitialized(ServletContextEvent sce) {
       ServletContext ctx = sce.getServletContext();
       StartupStatus ss = StartupStatus.getBean(ctx);

       try {
           FreemarkerEmailFactory factory = new FreemarkerEmailFactory(ctx);
           ctx.setAttribute(ATTRIBUTE_NAME, factory);

           if (factory.isConfigured()) {
               ss.info(this, "The system is configured to "
                       + "send mail to users.");
           } else {
               ss.info(this, "Configuration parameters are missing: "
                       + "the system will not send mail to users.");
           }
       } catch (Exception e) {
           ss.warning(this,
                   "Failed to initialize FreemarkerEmailFactory. "
                           + "The system will not be able to send email "
                           + "to users.", e);
       }
   }
```

```
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        sce.getServletContext().removeAttribute(ATTRIBUTE_NAME);
    }
}
```

Note that the StartupManager treats ServletContextListeners just like you would expect from reading the Servlet 2.4 specification:

- Only one instance of the listener is created per JVM.
- The contextInitialized() method is called once when the system is starting.
- The contextDestroyed() method is called on that same instance when the system shuts down.

8.3.13 Enhancing Freemarker templates with DataGetters

8.3.13.1 Overview

It is possible for a Freemarker template to display data that is not normally provided to it.

You can create an RDF file that describes a custom `DataGetter` object, and associates it with the desired template. Each time that template is used, the `DataGetter` will be executed, and the data will be stored in a variable, so the template can display it.

This does not require changes to the Java code. You create the RDF file in your VIVO distribution directory and modify the template in your theme.

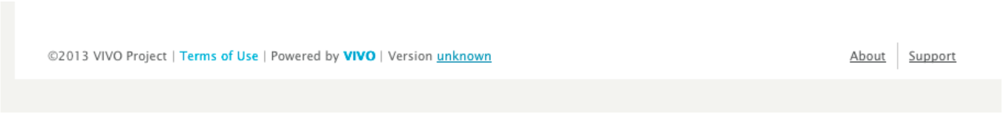
8.3.13.2 An example

Let's assume that we need to display information about the most recent data ingest operation. We want to display the name of the Person who supervised the ingest. We would like to display this on every page.

As part of the ingest process, we can load statements like this into the data model:

```
<http://vivo.mydomain.edu/individual/n5242>
    <http://vivo.mydomain.edu/individual/isMostRecentUpdater>
    "true" .
<http://vivo.mydomain.edu/individual/n5242>
    <http://www.w3.org/2000/01/rdf-schema#label>
    "Baker, Able" .
```

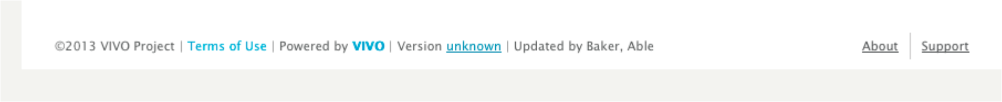
We would like for VIVO to display the name of this individual on every page, so the footer will change from this:



©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown

About | Support

to this:



©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown | Updated by Baker, Able

8.3.13.3 Creating the DataGetter

VIVO allows you to define and use `DataGetter` objects in several contexts. `DataGetter` s come in many flavors, but the most commonly used is the `SparqlQueryDataGetter` , which lets you define a SPARQL query, and store the results of that query for your Freemarker template to display.

By adding statements to your data model, you can define a `SparqlQueryDataGetter` object, and associate it with a Freemarker template. Here is the definition that is used in this example:

```
@prefix display: <http://vitro.mannlib.cornell.edu/ontologies/display/1.1#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<freemarker:footer.ftl> display:hasDataGetter display:updatedInfoDataGetter .

display:updatedInfoDataGetter
    a
<java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter> ;
    display:saveToVar "updatedInfo" ;
    display:query """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX local: <http://vivo.mydomain.edu/individual/>

    SELECT (str(?rawLabel) AS ?updater)
    WHERE {
        ?uri local:isMostRecentUpdater ?o ;
             rdfs:label ?rawLabel .
    }
    LIMIT 1
     """ .
```

These statements can be added to your data model in any of several ways. For this example, I stored these lines in a file called `data_getter_for_example.n3` and placed it in the VIVO distribution directory under `rdf/display/everytime` . Files placed in this directory are loaded when VIVO starts, but are not persisted when VIVO stops. This allows you to edit or remove the file without leaving residual statements in your data model.

Notice that:

- The first statement says that the Freemarker template `footer.ftl` has a `DataGetter` , defined in subsequent lines.
- The definition of the `DataGetter` states:
    - the type of the data getter,
    - the SPARQL query that will be executed

- the Freemarker variable that will hold the results.

The results of the SPARQL query are stored in a Freemarker variable, in this case `updatedInfo`. The variable will contain a Sequence of Hashes, where each Hash represents one line of the SPARQL result. Within each Hash, result values are specified as key/value pairs.

For more information on Sequences and Hashes, consult the Freemarker manual:

- Retrieving data from a Sequence[118]
- Retrieving data from a Hash[119]

8.3.13.4 Modifying the template

Here is the standard `footer.ftl` template:

footer.ftl

```
<#-- $This file is distributed under the terms of the license in /doc/license.txt$ -->
</div> <!-- #wrapper-content -->
<footer role="contentinfo">
    <p class="copyright">
        <#if copyright??>
            <small>&copy;${copyright.year?c}
            <#if copyright.url??>
                <a href="${copyright.url}" title="${i18n().menu_copyright}">${copyright.text}</a>
            <#else>
                ${copyright.text}
            </#if>
             | <a class="terms" href="${urls.termsOfUse}" title="${i18n().menu_termuse}">${i18n().menu_termuse}</a></small>
        </#if>
        ${i18n().menu_powered} <a class="powered-by-vivo" href="http://vivoweb.org" target="_blank" title="${i18n().menu_powered} VIVO"><strong>VIVO</strong></a>
        <#if user.hasRevisionInfoAccess>
             | ${i18n().menu_version} <a href="${version.moreInfoUrl}" title="${i18n().menu_version}">${version.label}</a>
        </#if>
    </p>
    <nav role="navigation">
        <ul id="footer-nav" role="list">
            <li role="listitem"><a href="${urls.about}" title="${i18n().menu_about}">${i18n().menu_about}</a></li>
            <#if urls.contact??>
```

---

[118] http://freemarker.sourceforge.net/docs/dgui_template_exp.html#dgui_template_exp_var_sequence

[119] http://freemarker.sourceforge.net/docs/dgui_template_exp.html#dgui_template_exp_var_hash

```
23                  <li role="listitem"><a href="${urls.contact}" title="${i18n().menu_contactus}">${i18n().menu_contactus}</a></li>
24              </#if>
25              <li role="listitem"><a href="http://www.vivoweb.org/support" target="blank" title="${i18n().menu_support}">${i18n().menu_support}</a></li>
26          </ul>
27      </nav>
28  </footer>
29  <#include "scripts.ftl">
```

Insert these lines between lines 17 and 18:

```
        <#if (updatedInfo?first.updater)??>
            | Updated by ${updatedInfo?first.updater}
        </#if>
```

The SPARQL result is obtained and stored into the Freemarker variable `updatedInfo` each time the `footer.ftl` template is loaded for display. The name we want is in the first row of the SPARQL result, keyed to the name `updater`.

8.3.13.5 Summary

Enhancing Freemarker templates is one more way to use the VIVO `DataGetter` mechanism. When you associate a `DataGetter` with a Freemarker template, that `DataGetter` will be run each time the template is invoked. This is true whether the template is specified by the controller, or included in another template. You can modify the template to display the data from the `DataGetter`, but it is prudent to include an `<#if>` tag, so your template won't fail if the data is not found.

8.3.14 Enriching profile pages using SPARQL query DataGetters

8.3.14.1 Introduction

VIVO supports the development of SPARQL query data getters that can be associated with specific ontological classes. These data getters, in turn, can be accessed within Freemarker templates to provide richer content on VIVO profile pages. For example, the profile page for an academic department lists only the names of the faculty within that department and their titles, but with a SPARQL query data getter it is possible to extend the faculty information to display all of the faculty members' research areas.

8.3.14.2 The Steps and an Example

There are five mandatory steps involved in developing and implementing a class-specific SPARQL query data getter. In this wiki page we'll walk through an example and provide details on each of these steps.

1. Define the customization
2. Write the SPARQL query
3. Produce the N3 for the data getter
4. Create a Freemarker template
5. Incorporate the new template into the application

8.3.14.2.1 Step 1. Define the Customization

This first step might seem obvious but it's helpful to define as specifically as possible the change being made to VIVO. For our example, we'll use the one mentioned in the introduction. On academic department pages, we'll provide a list of all the faculty members' research areas and we'll display these beneath the department overview near the top of the page. In addition, we want the listed research areas to be links that will take us to a detail page that shows all of the faculty who have selected a given research area. This last requirement, being able to drill down to a details page, requires both an additional template and data getter, and so we'll need an optional sixth step: Create the Drill-down Page Using Page Management.

8.3.14.2.2 Step 2. Write the SPARQL Query

Having defined our requirements, we now need to write a query that will return the data we want -- specifically, the rdfs labels of the research areas and, because we want to be able to drill-down on these labels, the URI of the research areas. An obvious place to write and test a query is the SPARQL Query page that you can access from the Site Admin page. Here's our test query:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
SELECT DISTINCT (str(?researchAreaLabel) AS ?raLabel) ?ra
WHERE {
        <http://localhost:8080/individual/n2936> vivo:organizationForPosition ?posn .
        ?posn  vivo:positionForPerson ?person .
        ?person vivo:hasResearchArea ?ra .
        ?ra rdfs:label ?researchAreaLabel
}
ORDER BY ?raLabel
```

There are two points to note here. In line 3 of the query we convert the label variable to a string to prevent any duplicate labels from appearing; and in line 5 we use the specific URI for an academic department. This URI allows us to test the query, but it will have to be replaced by a "generic" subject in our next step.

8.3.14.2.3 Step 3. Produce the N3 for the Data Getter

Once the SPARQL query has been tested, we define the data getter using triples stored in a .N3 file. This file is then placed in the WEB-INF directory in the VIVO source code, as follows: `rdf/display/everytime /deptResearchAreas.n3`.

The N3 for our data getter consists of two parts: (1) the triple that associates our data getter with the AcademicDepartment class and (2) the triples that define the data getter itself. Here is the former:

```
@prefix display: <http://vitro.mannlib.cornell.edu/ontologies/display/1.1#> .

<http://vivoweb.org/ontology/core#AcademicDepartment> display:hasDataGetter
display:getResearchAreaDataGetter .
```

And here are the triples that define the `getResearchAreaDataGetter` data getter:

```
display:getResearchAreaDataGetter
    a <java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter>;
    display:saveToVar "researchAreaResults";
    display:query """
      PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
      PREFIX vivo: <http://vivoweb.org/ontology/core#>
      PREFIX afn:  <http://jena.hpl.hp.com/ARQ/function#>
      PREFIX foaf: <http://xmlns.com/foaf/0.1/>
      SELECT DISTINCT (str(?researchAreaLabel) AS ?raLabel) ?ra
      WHERE {
         ?individualURI vivo:relatedBy ?posn .
         ?posn a vivo:Position .
         ?posn  vivo:relates ?person .
         ?person a foaf:Person .
         ?person vivo:hasResearchArea ?ra .
         ?ra rdfs:label ?researchAreaLabel
      }
      ORDER BY ?raLabel
      """ .
```

Note that we have exchanged our specific department URI with the variable `?individualURI`. `?individualURI` is a "built-in" variable; that is, when the data getter is executed the value of this variable is set to the URI of the individual whose page is being loaded. So in our example, because we have associated the data getter with the AcademicDepartment class, when the IndividualController loads an academic department, the URI of that department gets set as the value of the `?individualURI` variable in our query.

Also note line 3 of the data getter:

```
display:saveToVar "researchAreaResults".
```

The "save to" variable `researchAreaResults` is what we use to access the query results in our template.

8.3.14.2.4 Step 4. Create a Freemarker Template

Now that we've created our data getter, `getResearchAreaDataGetter`, and have a "save to" variable with which to access the query results, we create a Freemarker template -- individual-dept-research-areas.ftl -- and use the <#list> function to loop through and display the results. The following markup is all that's needed in this new template.

```
<#if researchAreaResults?has_content>
    <h2 id="facultyResearchAreas" class="mainPropGroup">
        Faculty Research Areas}
    </h2>
    <ul id="individual-hasResearchArea" role="list">
        <#list researchAreaResults as resultRow>
            <li class="raLink">
                <a class="raLink" href="${urls.base}/deptResearchAreas?deptURI=$
{individual.uri}&raURI=${resultRow["ra"]}" title="research area">
                    ${resultRow["raLabel"]}
                </a>
            </li>
        </#list>
    </ul>
</#if>
```

In the very first line we check to ensure that the query actually produced results. If not, no markup of any kind gets rendered. Otherwise, we give the new template section a heading, define an unordered list (<ul>) to contain the research areas, and then loop through the results. Note that the research area labels are contained within an anchor tag (<a>) because we want to be able to use these as links to a list of the faculty members for each research area. The URL in the href attribute includes what looks like a servlet name, `/deptResearchAreas,` and two parameters: `deptURI` and `raURI`. The `deptURI` parameter is the URI of the department that has been loaded by the IndividualController, and this value is accessible through the template variable ${individual.uri}. The `raURI` parameter is the URI of the research area, the value of which is available in our query results. These parameters and the servlet name will be used to develop the drill-down page that lists the faculty members in a department that have an interest in a specific research area.

8.3.14.2.5 Step 5. Incorporate the New Template into the Application

Now that we have a template to display the list of research areas, we need to update the individual.ftl template to source in the new template. Since individual.ftl is used to render individuals of many different classes, we include an <#if> statement to ensure that the individual-dept-research-areas.ftl template only gets included when the individual being loaded is an AcademicDepartment:

```
<#if individual.mostSpecificTypes?seq_contains("Academic Department")>
    <#include "individual-dept-research-areas.ftl">
</#if>
```

8.3.14.2.6 Step 6. Create the Drill-down Page Using Page Management (optional)

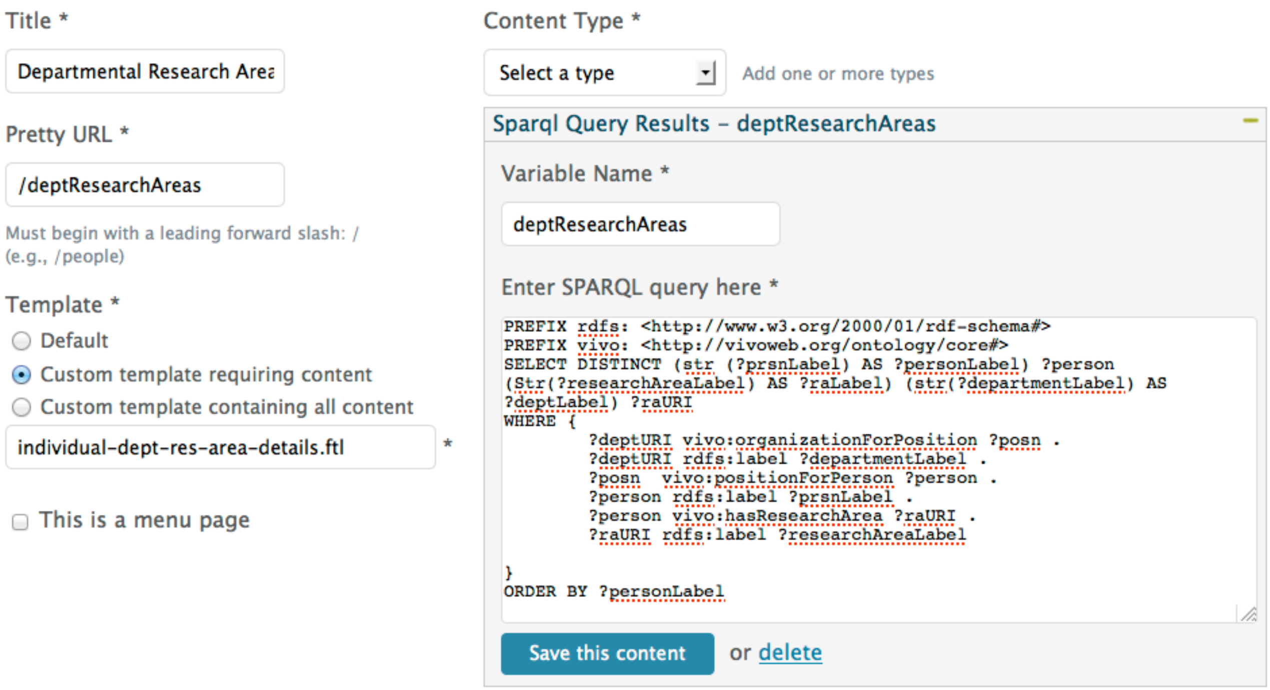
To this point, we have created a class-specific SPARQL query data getter, which retrieves the research areas of the faculty in a given academic department; developed a new template to render the results of our data getter; and updated the individual.ftl template to display the list of research areas. In Step 1, however, we defined requirements that include the ability to drill down from a selected research area to display a list of the faculty members in the department who have an interest in that research area. This is also done using a SPARQL query and new template. But in this case the query does not need to be associated with a specific class and defined in an .N3 file. Instead, we can create a SPARQL query page using the Page Management[120] functionality.

As noted in Step 4, the anchor tags in the list of research areas include an `href` attribute that takes this format:

```
href="${urls.base}/deptResearchAreas?deptURI=${individual.uri}&raURI=${resultRow["ra"
]}"
```

When creating the SPARQL query page in Page Management, as shown in the illustration below, we set the "Pretty URL" field to `/deptResearchAreas`. This portion of the `href` attribute, then, is not the name of an actual servlet but it effectively functions as one, and it is also associated with the template that we also define in Page Management: individual-dept-res-area-details.ftl. When a user clicks on one of the listed research areas, this is the template that the application will load.

Note the SPARQL query that is defined in the illustration below. It uses as variables the same parameters that are part of the `href` above: `deptURI` and `raURI`. Like the `?individualURI` discussed in Step 3, the values of these two parameters will become the values of the corresponding variables in the SPARQL query.



Title *: Departmental Research Area

Pretty URL *: /deptResearchAreas

Must begin with a leading forward slash: / (e.g., /people)

Template *:
○ Default
● Custom template requiring content
○ Custom template containing all content

individual-dept-res-area-details.ftl *

☐ This is a menu page

Content Type *

Sparql Query Results – deptResearchAreas

Variable Name *: deptResearchAreas

Enter SPARQL query here *

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
SELECT DISTINCT (str (?prsnLabel) AS ?personLabel) ?person
(Str(?researchAreaLabel) AS ?raLabel) (str(?departmentLabel) AS
?deptLabel) ?raURI
WHERE {
        ?deptURI vivo:organizationForPosition ?posn .
        ?deptURI rdfs:label ?departmentLabel .
        ?posn  vivo:positionForPerson ?person .
        ?person rdfs:label ?prsnLabel .
        ?person vivo:hasResearchArea ?raURI .
        ?raURI rdfs:label ?researchAreaLabel

}
ORDER BY ?personLabel
```

Now that the SPARQL query page has been created in Page Management, we still need to create the individual-dept-res-area-details.ftl template. Just as in Step 4, where we used the "save to" variable to access the query results in the individual-dept-research-areas.ftl template, we now use the variable defined in the "Variable Name" field (above) to access the results of that SPARQL query. Here is the content of the new template:

```
<#if deptResearchAreas?has_content>
```

---

[120] https://wiki.duraspace.org/display/VIVO/Customize%3A+Page+management

```
    <section id="pageList">
        <#list deptResearchAreas as firstRow>
            <div class="tab">
                <h2>${firstRow["raLabel"]}</h2>
                <p>
                    Here are the faculty members in the ${firstRow["deptLabel"]}
department with an interest in this research area.
                </p>
            </div>
            <#break>
        </#list>
    </section>
    <section id="deptResearchAreas">
        <ul role="list" class="deptDetailsList">
            <#list deptResearchAreas as resultRow>
                <li class="deptDetailsListItem">
                    <a href="${urls.base}/individual${resultRow["person"]?
substring(resultRow["person"]?last_index_of("/"))}"
                    title="faculty name">
                        ${resultRow["personLabel"]}
                    </a>
                </li>
            </#list>
        </ul>
    </section>
</#if>
```

Once again we use an <#if> statement to check for results. But this time we use the <#list> function twice: once to retrieve just the first row, which is used to provide a heading and some introductory text; and a second time to list all of the faculty members with an interest in the selected research area.

8.3.15 Multiple profile types for foaf:Person

8.3.15.1 Introduction

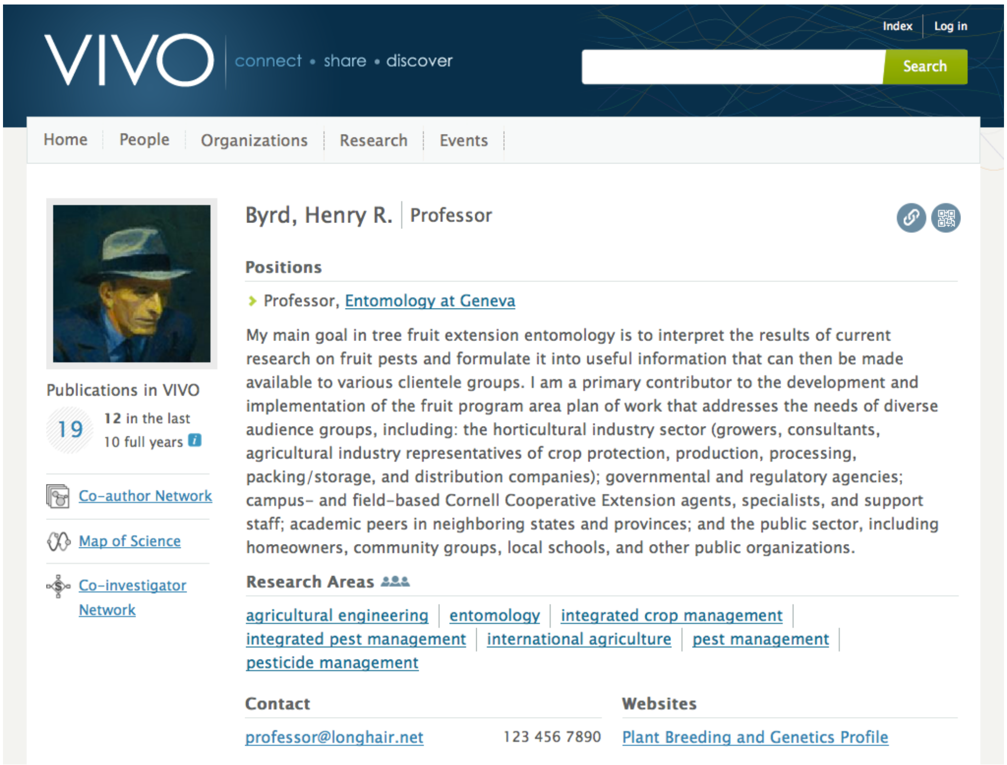
VIVO now supports multiple profile pages for foaf:Persons. This feature, which is optional so installations can continue to use just the individual--foaf-person.ftl template, currently consists of two profile page types: a standard view, which is a redesigned version of the foaf:Person template in previous releases; and a quick view, which emphasizes the individual's own web page presence while providing summary VIVO information, such as current positions and research areas. The profile quick view requires the use of a web service that captures images of web pages. This web service is not included with the VIVO software, so an installation will either have to develop their own service or use a third-party service, usually for a small fee depending on the number of images served. Examples of these services include WebShotsPro, Thumbalizr and Websnapr.

8.3.15.2 The Profile Page Types

As noted above, there are currently two supported profile page types. Here are examples of those two views

8.3.15.2.1 The Standard View

The standard view is similar to the default foaf:Person template except that the information displayed at the top of the page is divided into only two primary columns instead of three. The actual template name for this page type is individual--foaf-person-2column.ftl.



Byrd, Henry R. | Professor

Positions

- Professor, Entomology at Geneva

My main goal in tree fruit extension entomology is to interpret the results of current research on fruit pests and formulate it into useful information that can then be made available to various clientele groups. I am a primary contributor to the development and implementation of the fruit program area plan of work that addresses the needs of diverse audience groups, including: the horticultural industry sector (growers, consultants, agricultural industry representatives of crop protection, production, processing, packing/storage, and distribution companies); governmental and regulatory agencies; campus- and field-based Cornell Cooperative Extension agents, specialists, and support staff; academic peers in neighboring states and provinces; and the public sector, including homeowners, community groups, local schools, and other public organizations.

Research Areas

agricultural engineering | entomology | integrated crop management | integrated pest management | international agriculture | pest management | pesticide management

Contact

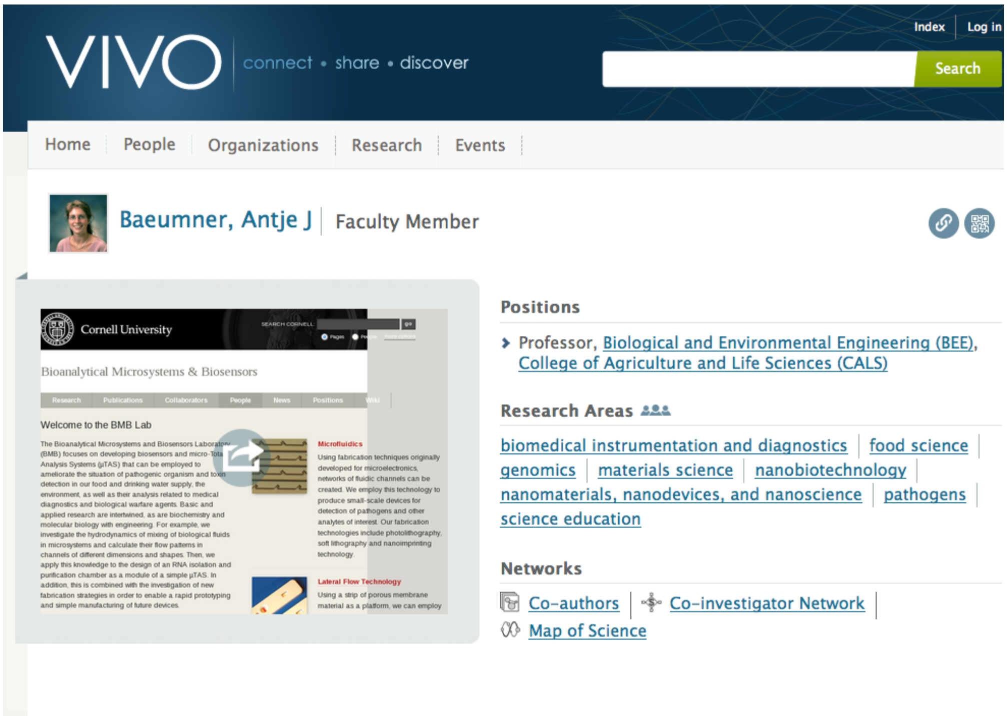
professor@longhair.net 123 456 7890

Websites

Plant Breeding and Genetics Profile

8.3.15.2.2 The Quick View

As illustrated below, the quick view puts a visual emphasis on the individual's own web presence. In this case, the person only has one web page displayed. When there is more than one, the primary web page is displayed as shown and any additional web pages are displayed as thumbnails beneath the primary one.


VIVO
connect • share • discover
Index
Log in
Search
Home
People
Organizations
Research
Events
Cornell University
Bioanalytical Microsystems & Biosensors
Welcome to the BMB Lab
Microfluidics
Lateral Flow Technology

Baeumner, Antje J | Faculty Member

Positions

- Professor, Biological and Environmental Engineering (BEE), College of Agriculture and Life Sciences (CALS)

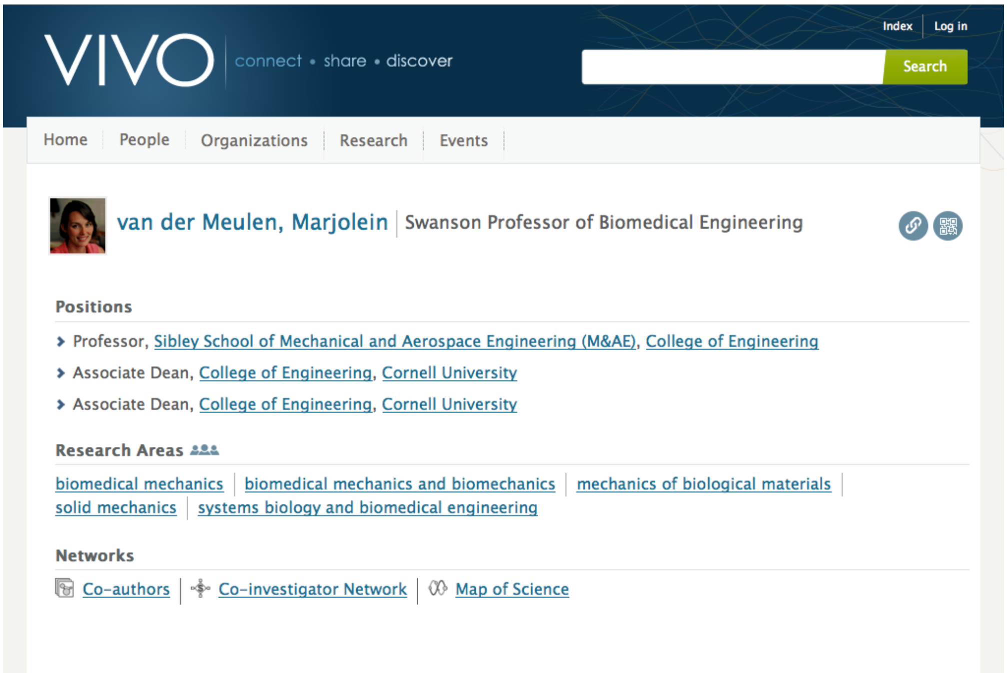
Research Areas

biomedical instrumentation and diagnostics | food science | genomics | materials science | nanobiotechnology | nanomaterials, nanodevices, and nanoscience | pathogens | science education

Networks

Co-authors | Co-investigator Network | Map of Science

It's possible that there will be some individuals who do not have a web page to display. In that situation the quick view will display as follows.


VIVO
connect • share • discover
Index
Log in
Search
Home
People
Organizations
Research
Events

van der Meulen, Marjolein | Swanson Professor of Biomedical Engineering

Positions

- Professor, Sibley School of Mechanical and Aerospace Engineering (M&AE), College of Engineering
- Associate Dean, College of Engineering, Cornell University
- Associate Dean, College of Engineering, Cornell University

Research Areas

biomedical mechanics | biomedical mechanics and biomechanics | mechanics of biological materials | solid mechanics | systems biology and biomedical engineering

Networks

Co-authors | Co-investigator Network | Map of Science

8.3.15.3 Implementing Multiple Profile Pages

Here are the steps required to implement the multiple profile pages feature.

1. Develop or a website image capture service
2. Update the runtime.properties file
3. Override the default foaf:Person template
4. Update the webpage quick view template
5. Set the Profile Page Type for your foaf:Persons

8.3.15.3.1 Step 1. Develop or a Website Image Capture Service

Since there are currently only two page views, and one of those emphasizes the individual's own web site, to implement the multiple profile pages feature requires that an installation either develop its own web service for capturing images of web sites or select a third-party service for this purpose. As noted in the introduction, these services include WebShotsPro, Thumbalizr and Websnapr.

A third option, however, would be to modify the quick view template (individual--foaf-person-quickview.ftl) so that it does not display a web page image (as in the third screen shot above). This template file is located in the `productMods/templates/freemarker/body/individual` directory.

8.3.15.3.2 Step 2. Update the runtime.properties File

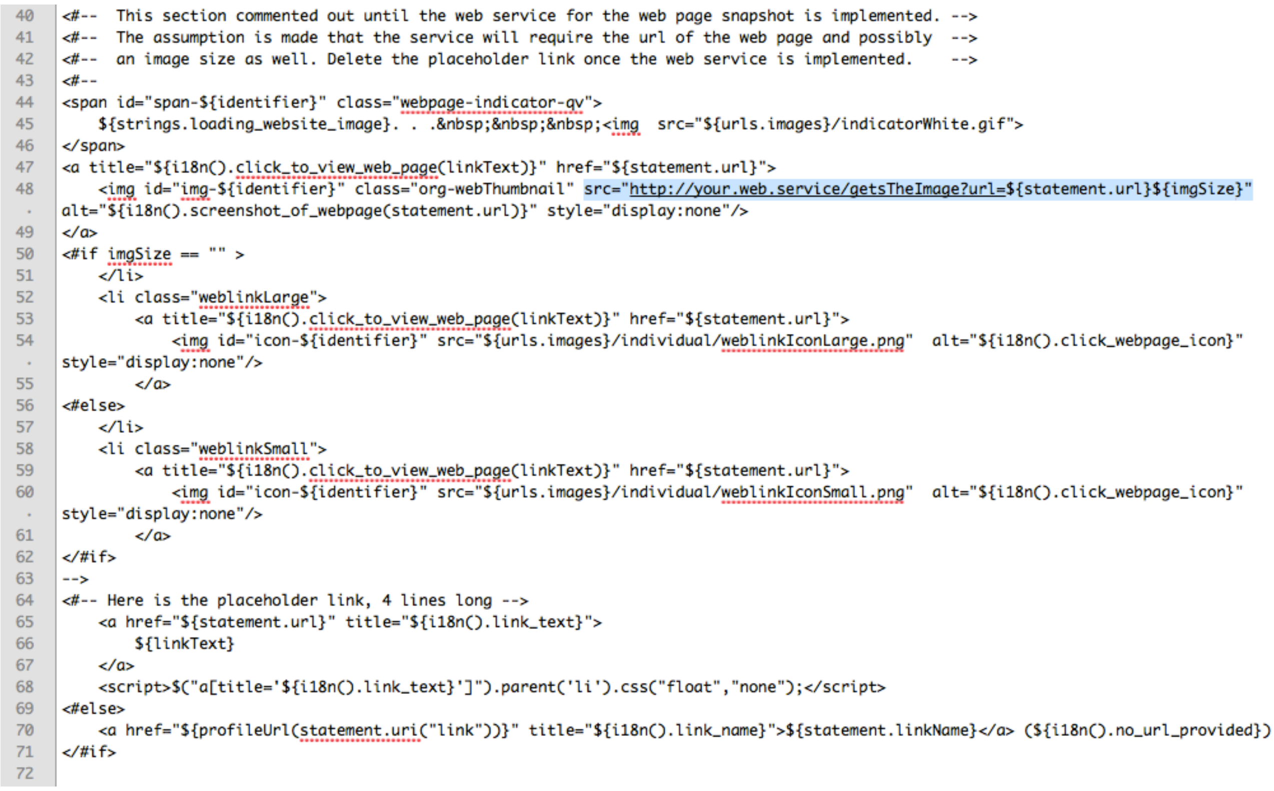
Set the multiViews.profilePageTypes to "enabled" and ensure that it is not commented out.

8.3.15.3.3 Step 3. Override the Default foaf:Person Template

There are two ways to override the default individual--foaf-person.ftl template, which is located in the `themes/wilma/templates` directory: (1) rename the file, or (2) remove it from that directory.

8.3.15.3.4 Step 4. Update the Webpage Quick View Template

The template that displays the web page image in the quick view is named propStatement-webpage-quickview.ftl. As delivered, this template uses a placeholder link (or links) to display the individual's web page (or pages), while the code that calls the web service is currently commented out. Here is that section of the template:

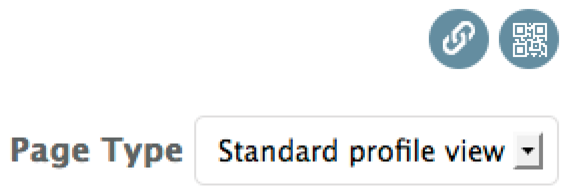


```
40  <#--  This section commented out until the web service for the web page snapshot is implemented. -->
41  <#--  The assumption is made that the service will require the url of the web page and possibly  -->
42  <#--  an image size as well. Delete the placeholder link once the web service is implemented.    -->
43  <#--
44  <span id="span-${identifier}" class="webpage-indicator-qv">
45      ${strings.loading_website_image}. . .&nbsp;&nbsp;&nbsp;<img  src="${urls.images}/indicatorWhite.gif">
46  </span>
47  <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
48      <img id="img-${identifier}" class="org-webThumbnail" src="http://your.web.service/getsTheImage?url=${statement.url}${imgSize}"
 ·  alt="${i18n().screenshot_of_webpage(statement.url)}" style="display:none"/>
49  </a>
50  <#if imgSize == "" >
51      </li>
52      <li class="weblinkLarge">
53          <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
54              <img id="icon-${identifier}" src="${urls.images}/individual/weblinkIconLarge.png"  alt="${i18n().click_webpage_icon}"
 ·  style="display:none"/>
55          </a>
56  <#else>
57      </li>
58      <li class="weblinkSmall">
59          <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
60              <img id="icon-${identifier}" src="${urls.images}/individual/weblinkIconSmall.png"  alt="${i18n().click_webpage_icon}"
 ·  style="display:none"/>
61          </a>
62  </#if>
63  -->
64  <#-- Here is the placeholder link, 4 lines long -->
65      <a href="${statement.url}" title="${i18n().link_text}">
66          ${linkText}
67      </a>
68      <script>$("a[title='${i18n().link_text}']").parent('li').css("float","none");</script>
69  <#else>
70      <a href="${profileUrl(statement.uri("link"))}" title="${i18n().link_name}">${statement.linkName}</a> (${i18n().no_url_provided})
71  </#if>
72
```

Note the highlighted text on line 48. The URL in the src attribute is where you call either the web service you developed or the third-party service. The APIs for these services are fairly standard. Besides the URL of the web site that will be the source of the screen shot, the code in this template assumes that the API also takes an image size. For example, some services can provide small, medium and large images; others may only provide a large image and a thumbnail image. Once you've updated line 48 to call your web service, remember to comment out or remove the placeholder link, lines 65-68.

8.3.15.3.5 Step 5. Set the Profile Page Type for your foaf:Persons

When multiple profile pages are implemented, the default view is the standard profile view. You can change an individual's profile page type through the GUI by accessing the Page Type drop down:



Page Type: Standard profile view

You can also set the profile page type by ingesting RDF. An N3 triple, for example would consist of the following parts:

- the subject would be the URI of the individual, such as

  `<http://localhost:8080/individual/n7829>` ;

- the predicate would be the hasDefaultProfilePageType object property,

  `<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#hasDefaultProfilePageType>` ;
- and the object would be the type of profile,

  `<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#quickView>` (or `#standard`).

The ProfilePageType class is defined in the display model. Refer to the profilePageType.n3 file for details.

8.3.15.4 Using the Standard View Without Implementing Multiple Profile Pages

It's possible that an installation may want to use the standard view instead of the default foaf:Person template, but does not want to implement multiple profile pages. This can be done by simply (1) overriding the default foaf:Person template (just as in Step 3 above) and (2) ensuring that the multiViews.profilePageTypes properties in the runtime.properties file is either commented out or set to "disabled."

8.3.16 Using OpenSocial Gadgets

8.3.16.1 Overview

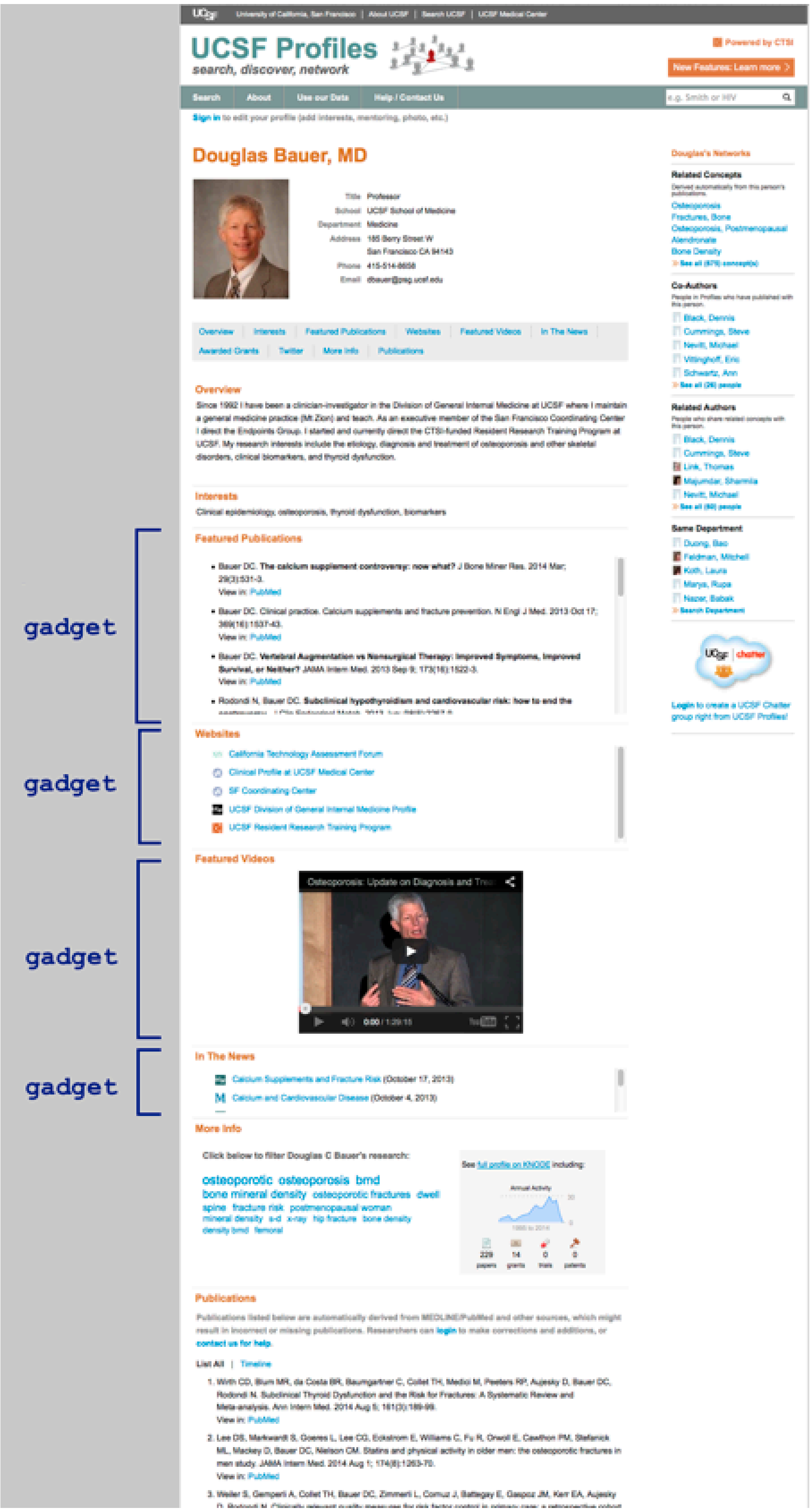
8.3.16.1.1 What can you do?

Your site administrators can configure a collection of "gadgets" for your VIVO installation. From that collection, each faculty member can decide which gadgets he will show on his profile page, and how they should be configured.

Perhaps it would be better to describe the gadgets as "page sections", because you can use CSS styling to make the gadget seamlessly match your theme. The result is profile pages that still look unified, but are at least partially configurable by the individual faculty member.

8.3.16.1.2 An example

Here is a portion of a profile page from UCSF Profiles. Each gadget is there because the page owner selected it and configured it.



Sign in to edit your profile (add interests, mentoring, photo, etc.)

Douglas Bauer, MD

Title: Professor
School: UCSF School of Medicine
Department: Medicine
Address: 185 Berry Street W
San Francisco CA 94143
Phone: 415-514-8658
Email: dbauer@psg.ucsf.edu

Overview

Since 1992 I have been a clinician-investigator in the Division of General Internal Medicine at UCSF where I maintain a general medicine practice (Mt Zion) and teach. As an executive member of the San Francisco Coordinating Center I direct the Endpoints Group. I started and currently direct the CTSI-funded Resident Research Training Program at UCSF. My research interests include the etiology, diagnosis and treatment of osteoporosis and other skeletal disorders, clinical biomarkers, and thyroid dysfunction.

Interests

Clinical epidemiology, osteoporosis, thyroid dysfunction, biomarkers

Featured Publications

- Bauer DC. **The calcium supplement controversy: now what?** J Bone Miner Res. 2014 Mar; 29(3):531-3.
  View in: PubMed
- Bauer DC. Clinical practice. Calcium supplements and fracture prevention. N Engl J Med. 2013 Oct 17; 369(16):1537-43.
  View in: PubMed
- Bauer DC. **Vertebral Augmentation vs Nonsurgical Therapy: Improved Symptoms, Improved Survival, or Neither?** JAMA Intern Med. 2013 Sep 9; 173(16):1522-3.
  View in: PubMed
- Rodondi N, Bauer DC. **Subclinical hypothyroidism and cardiovascular risk: how to end the**

Websites

- California Technology Assessment Forum
- Clinical Profile at UCSF Medical Center
- SF Coordinating Center
- UCSF Division of General Internal Medicine Profile
- UCSF Resident Research Training Program

Featured Videos

Osteoporosis: Update on Diagnosis and Tre
0:00 / 1:29:15

In The News

- Calcium Supplements and Fracture Risk (October 17, 2013)
- Calcium and Cardiovascular Disease (October 4, 2013)

More Info

Click below to filter Douglas C Bauer's research:

osteoporotic osteoporosis bmd bone mineral density osteoporotic fractures dwell spine fracture risk postmenopausal woman mineral density s-d x-ray hip fracture bone density density bmd femoral

See full profile on KNODE including:

Annual Activity
30
0
1985 to 2014

229	14	0	0
papers	grants	trials	patents

Publications

Publications listed below are automatically derived from MEDLINE/PubMed and other sources, which might result in incorrect or missing publications. Researchers can **login** to make corrections and additions, or **contact us for help**.

List All | Timeline

1. With CD, Blum MR, da Costa BR, Baumgartner C, Collet TH, Medici M, Peeters RP, Aujesky D, Bauer DC, Rodondi N. Subclinical Thyroid Dysfunction and the Risk for Fractures: A Systematic Review and Meta-analysis. Ann Intern Med. 2014 Aug 5; 161(3):189-99.
   View in: PubMed
2. Lee DS, Markwardt S, Goeres L, Lee CG, Eckstrom E, Williams C, Fu R, Orwoll E, Cawthon PM, Stefanick ML, Mackey D, Bauer DC, Nielson CM. Statins and physical activity in older men: the osteoporotic fractures in men study. JAMA Intern Med. 2014 Aug 1; 174(8):1263-70.
   View in: PubMed
3. Weiler S, Gemperli A, Collet TH, Bauer DC, Zimmerli L, Cornuz J, Battegay E, Gaspoz JM, Kerr EA, Aujesky

Douglas's Networks

Related Concepts
Derived automatically from this person's publications.
- Osteoporosis
- Fractures, Bone
- Osteoporosis, Postmenopausal
- Alendronate
- Bone Density
- See all (679) concept(s)

Co-Authors
People in Profiles who have published with this person.
- Black, Dennis
- Cummings, Steve
- Nevitt, Michael
- Vittinghoff, Eric
- Schwartz, Ann
- See all (26) people

Related Authors
People who share related concepts with this person.
- Black, Dennis
- Cummings, Steve
- Link, Thomas
- Majumdar, Sharmila
- Nevitt, Michael
- See all (60) people

Same Department
- Duong, Bao
- Feldman, Mitchell
- Koth, Laura
- Marya, Rupa
- Nazer, Babak
- Search Department

UCSF | chatter

Login to create a UCSF Chatter group right from UCSF Profiles!

8.3.16.1.3 OpenSocial

The OpenSocial standard was developed to make it easy for developers to add functionality to social networking systems like Google and MySpace. OpenSocial has lost popularity in social networking, but is becoming more favored in enterprise systems.

The Clinical and Translational Science Institute at UCSF created a project to host OpenSocial gadgets in the Harvard Profiles system. In keeping with the cross-platform origins of OpenSocial, the CTSI team decided to adapt their gadgets for use in VIVO as well.

8.3.16.1.4 ORNG

Social networking systems provide very little information about their participants. The group at CTSI wanted to combine the display tools of OpenSocial with the data structure of VIVO RDF. They accomplished this through an extension to the standard, which they called Open Research Networking Gadgets, or ORNG.

8.3.16.2 Adding gadgets to VIVO

The gadgets used at UCSF are provided in a library. Some are written specifically for UCSF, or specifically for the Harvard Profiles platform. However, many are available for use in VIVO.

You can also create your own gadgets. The gadgets are written in JavaScript, and you can use the existing gadgets as coding examples.

8.3.16.2.1 Under your control

The VIVO administrators select which gadgets will be available for the site. They also decide where the gadgets will appear on a profile page, if enabled.

8.3.16.2.2 Under control of your faculty

Each page owner may choose to enable individual gadgets for their page. A gadget may be written to accept settings that allow further configuration of its content and appearance.

8.3.16.3 Getting started

The VIVO Installation Instructions contain a section on how to add OpenSocial gadgets to a VIVO site. This will require some setup, and re-deploying VIVO. Once those steps are completed, your gadget library is configured by settings in a MySQL database table, and the gadget appearance is controlled by your Freemarker templates and CSS files.

For more information about Open Research Networking Gadgets, see the ORNG web site[121].

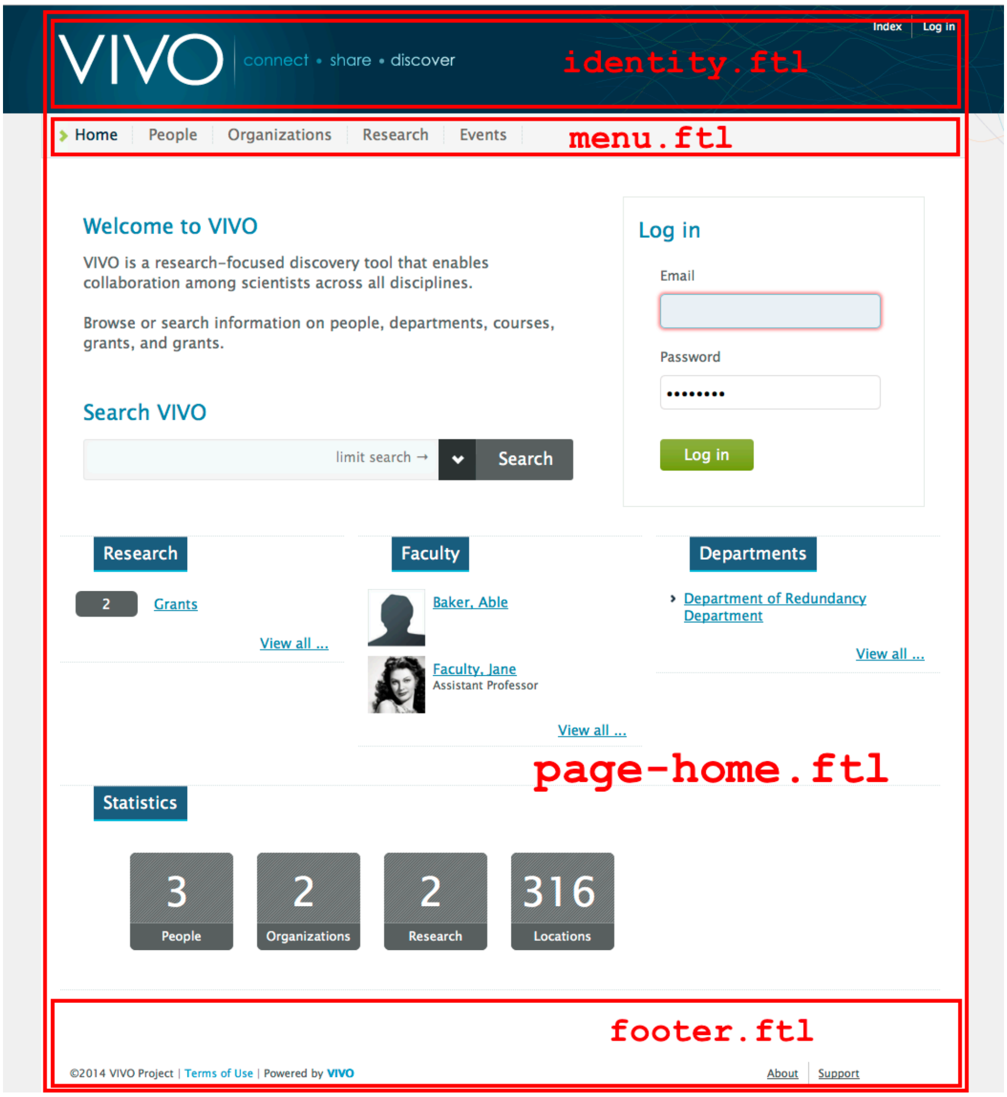
---

[121] http://www.orng.info/

8.3.17 How VIVO creates a page

8.3.17.1 The home page

Like the title page of a book, it is not unusual for the home page of a web site to be different from all other pages. In the default VIVO theme, the most significant difference is that the search box is moved from the header to a more prominent location on the page.


VIVO
connect • share • discover
Index
Log in
identity.ftl
Home
People
Organizations
Research
Events
menu.ftl

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and grants.

Search VIVO
limit search →
Search
Log in
Email
Password
••••••••
Log in
Research
2
Grants
View all ...
Faculty
Baker, Able
Faculty, Jane
Assistant Professor
View all ...
Departments
Department of Redundancy Department
View all ...
page-home.ftl
Statistics
3
People
2
Organizations
2
Research
316
Locations
footer.ftl
About
Support

©2014 VIVO Project | Terms of Use | Powered by VIVO

The following templates are used in the home page.

```
pageSetup.ftl
page-home.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
```

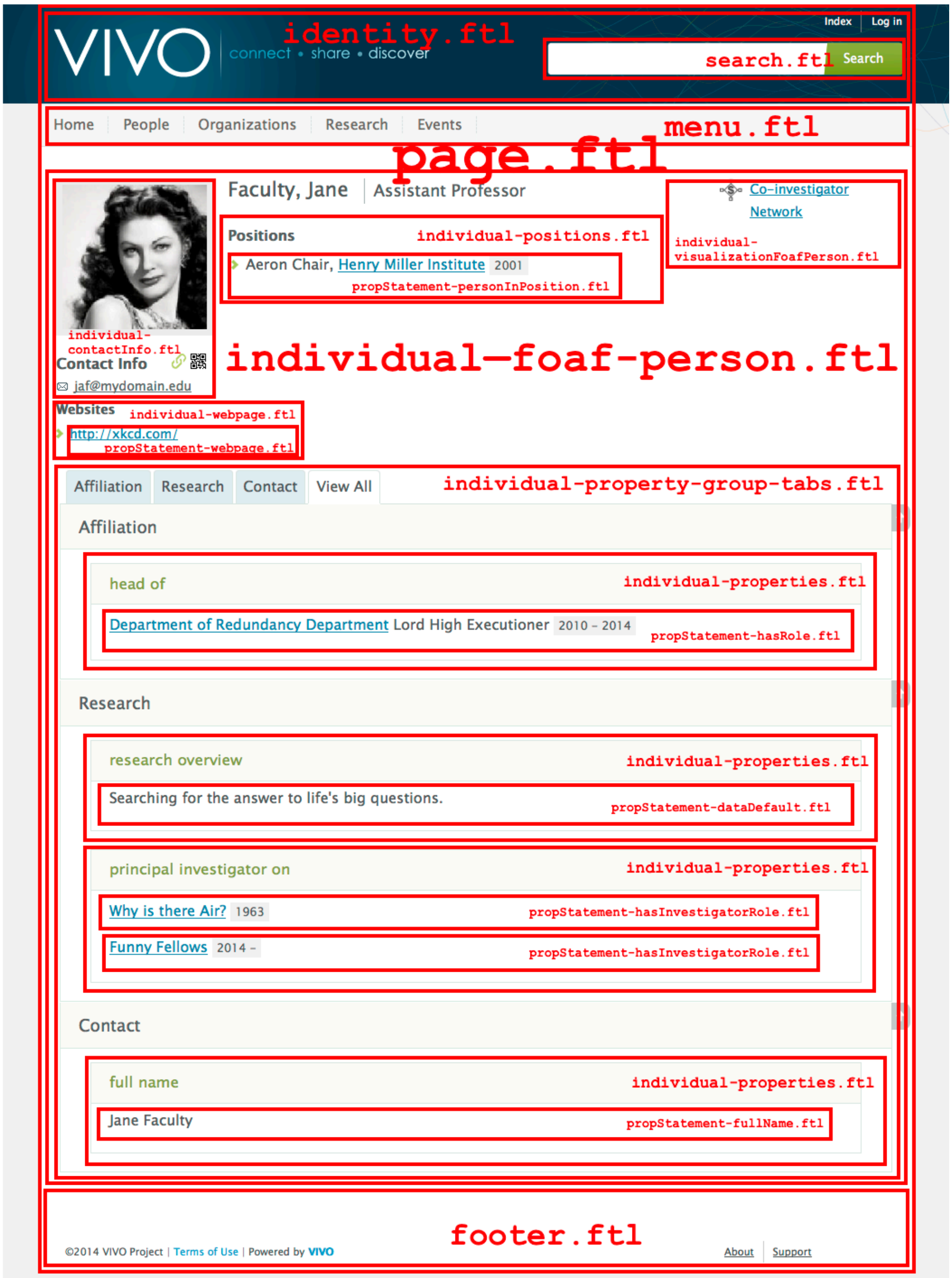
```
  identity.ftl
    languageSelector.ftl
  menu.ftl
    developer.ftl
  footer.ftl
    scripts.ftl
    googleAnalytics.ftl
```

Template	Purpose	From
`pageSetup.ftl`	Sets some class and formatting parameters.	Included in every page, by `TemplateProcessingHelper.java`
`page-home.ftl`	The special template used for the home page.	Specified as the page template by `HomePageController.java`, overriding the default `page.ftl`.
`head.ftl`	Creates the HTML `<HEAD>` tag.	Included by `page-home.ftl`.
`stylesheets.ftl`	Inserts links to CSS stylesheets.	Included by `head.ftl`.
`headScripts.ftl`	Inserts links to JavaScript files that must appear in the `<HEAD>` section of the page. These are somewhat unusual, since most JavaScript links appear at the end of the page.	Included by `head.ftl`.
`identity.ftl`	Draws the heading of the heading of the page, including the VIVO logo and the `Index` and `Log in` links.	Included by `page-home.ftl`.
`languageSelector.ftl`	Allows the user to select their preferred language. If the site supports only one language, this template has no effect.	Included by `identity.ftl`.

Template	Purpose	From
`menu.ftl`	Displays the page links (`Home` , `People` , etc.) at the top of the page.	Included by `page-home.ftl` .
`developer.ftl`	Displays the developer panel, used when testing and monitoring VIVO operation. If developer mode has not been enabled, this templates produces nothing.	Included by `menu.ftl` .
`footer.ftl`	Draws the footer of the page, including the copyright notice, and the `About` and `Support` links.	Included by `page-home.ftl` .
`scripts.ftl`	Inserts links to JavaScript files. Compare to `headScripts.ftl` .	Included by `footer.ftl` .
`googleAnalytics.ftl`	Inserts JavaScript code to work with Google Analytics. By default, this is commented out, since each site will need to insert their own ID values in order to produce meaningful results.	Included by `footer.ftl` .

8.3.17.2 A profile page

By numbers, the vast majority of pages on a VIVO site are profile pages. These are all likely to be structured around the properties of each individual. However, the format can be very different depending on whether that individual is a person, an organization, or a research grant.


VIVO
connect • share • discover
identity.ftl
Index
Log in
search.ftl
Search
Home
People
Organizations
Research
Events
menu.ftl
page.ftl
Faculty, Jane
Assistant Professor
Co-investigator Network
individual-visualizationFoafPerson.ftl
Positions
individual-positions.ftl
Aeron Chair, Henry Miller Institute 2001
propStatement-personInPosition.ftl
individual-contactInfo.ftl
Contact Info
jaf@mydomain.edu
individual–foaf-person.ftl
Websites
individual-webpage.ftl
http://xkcd.com/
propStatement-webpage.ftl
Affiliation
Research
Contact
View All
individual-property-group-tabs.ftl
Affiliation
head of
individual-properties.ftl
Department of Redundancy Department Lord High Executioner 2010 - 2014
propStatement-hasRole.ftl
Research
research overview
individual-properties.ftl
Searching for the answer to life's big questions.
propStatement-dataDefault.ftl
principal investigator on
individual-properties.ftl
Why is there Air? 1963
propStatement-hasInvestigatorRole.ftl
Funny Fellows 2014 -
propStatement-hasInvestigatorRole.ftl
Contact
full name
individual-properties.ftl
Jane Faculty
propStatement-fullName.ftl
footer.ftl
©2014 VIVO Project | Terms of Use | Powered by VIVO
About
Support

The following templates are used in this particular profile page. As explained in the notes, the choice of templates is driven in part by the content of the page.

```
pageSetup.ftl
page.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
```

```
            languageSelector.ftl
        search.ftl
        menu.ftl
            developer.ftl
        individual--foaf-person.ftl
            individual-setup.ftl
            individual-orcidInterface.ftl
            individual-contactInfo.ftl
            individual-webpage.ftl
                propStatement-webpage.ftl
            individual-visualizationFoafPerson.ftl
            individual-adminPanel.ftl
            individual-positions.ftl
                propStatement-personInPosition.ftl
            individual-overview.ftl
            individual-researchAreas.ftl
            individual-geographicFocus.ftl
        individual-openSocial.ftl
        individual-property-group-tabs.ftl
            individual-properties.ftl
                propStatement-hasRole.ftl
            individual-properties.ftl
                propStatement-dataDefault.ftl
                propStatement-hasInvestigatorRole.ftl
                propStatement-hasInvestigatorRole.ftl
            individual-properties.ftl
                propStatement-fullName.ftl
        footer.ftl
            scripts.ftl
            googleAnalytics.ftl
```

Template	Purpose	From
`pageSetup.ftl`	*as above.*	
`page.ftl`	The master template for most VIVO pages.	Specified by `FreemarkerHttpServlet.java`.

`head.ftl`
`    stylesheets.ftl`
`    headScripts.ftl`
`identity.ftl`
 	
`languageSelector.ftl`
	as above.	

Template	Purpose	From
`search.ftl`	Draws the search box in the header of the page.	Included by `page.ftl`.
`menu.ftl` `developer.ftl`	*as above.*	
`individual--foaf-person.ftl`	The main body of the profile page.	VIVO is configured to use this template as the body of a profile page for any `foaf:Person`. **You can change this configuration: see** Class-specific templates for profile pages (see page 183)**.** This is specified in `initialTBoxAnnotations.n3`, and recognized by `IndividualResponseBuilder.java` and `IndividualTemplateLocator.java`.
`individual-setup.ftl`	Sets some basic values for the following templates to use.	Included by `individual--foaf-person.ftl`.
`individual-orcidInterface.ftl`	Implements the VIVO integration to ORCiD. If this integration is not enabled, this template has no effect.	Included by `individual--foaf-person.ftl`.
`individual-contactInfo.ftl`	Displays the person's phone numbers and email addresses.	Included by `individual--foaf-person.ftl`.
`individual-webpage.ftl`	Displays the person's preferred web pages.	Included by `individual--foaf-person.ftl`.

Template	Purpose	From
`propStatement-webpage.ftl`	Displays a link to one of the person's preferred web pages.	VIVO is configured to use this template when displaying preferred web pages. **You can change this configuration: see** Custom List View Configurations (see page 188). This is specified in `listViewConfig-webpage.xml`, which is specified in `PropertyConfig.n3` and `vivoListViewConfig.rdf`.
`individual-visualizationFoafPerson.ftl`	Displays the visualization links for co-authors, co-investigators	Included by `individual--foaf-person.ftl`.
`individual-adminPanel.ftl`	Displays links for a VIVO administrator to use when editing this person's information	Included by `individual--foaf-person.ftl`.
`individual-positions.ftl`	Displays the positions that this person currently holds.	Included by `individual--foaf-person.ftl`.
`propStatement-personInPosition.ftl`	Displays one position that this person currently holds.	VIVO is configured to use this template when displaying positions. **You can change this configuration: see** Custom List View Configurations (see page 188). This is specified in `listViewConfig-personInPosition.xml`, which is specified in `PropertyConfig.n3`.

`individual-overview.ftl`
`individual-researchAreas.ftl`
`individual-geographicFocus.ftl`
	Display additional information about the person.	Included by `individual--foaf-person.ftl`.

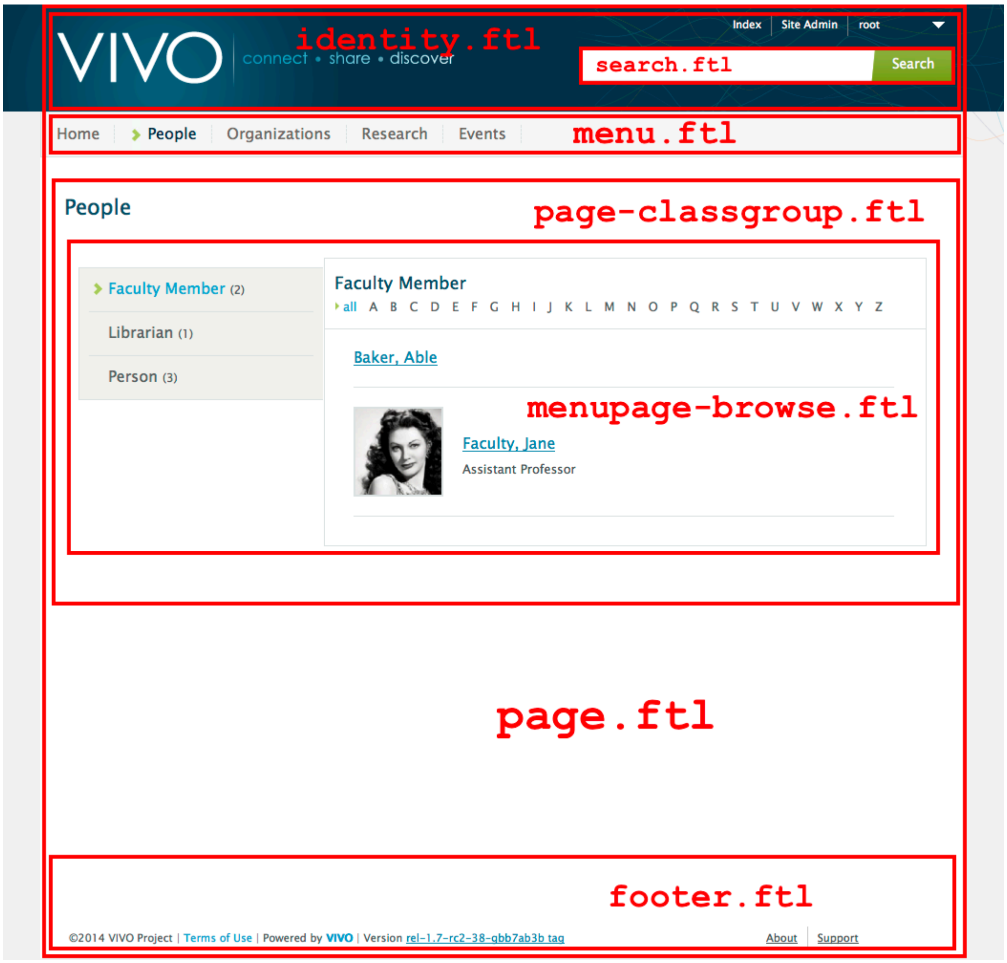
Template	Purpose	From
`individual-openSocial.ftl`	Implements the VIVO integration to OpenSocial gadgets. If this integration is not enabled, this template has no effect.	**You can configure VIVO to display OpenSocial gadgets on profile pages: see** Using OpenSocial Gadgets (see page 245)**.** Included by `individual--foaf-person.ftl`.
`individual-property-group-tabs.ftl`	Displays the groups of properties for this person.	Included by `individual--foaf-person.ftl`.

`individual-properties.ftl`
`propStatement-hasRole.ftl`
`individual-properties.ftl`
`propStatement-dataDefault.ftl`
`propStatement-hasInvestigatorRole.ftl`
`propStatement-hasInvestigatorRole.ftl`
`individual-properties.ftl`
`propStatement-fullName.ftl`
	Each invokation of `individual-properties.ftl` displays a property group.
	Each subordinate template displays one property for this person.
		Each reference to `individual-properties.ftl` is included by `individual-property-group-tabs.ftl`.
		VIVO is configured to use these subordinate templates when displaying research overview, roles, and names. **You can change this configuration: see** Custom List View Configurations (see page 188)**.**

`footer.ftl`
`scripts.ftl`
`googleAnalytics.ftl`
	as above.	

8.3.17.3 The People page

The page management GUI provides an easy way for VIVO administrators to create simple pages. These pages may also be added to the menu bar. By default, VIVO is configured with eleven such pages. Five of them are listed in the menu.


VIVO
connect • share • discover
identity.ftl
Index
Site Admin
root
search.ftl
Search
Home
People
Organizations
Research
Events
menu.ftl
People
page-classgroup.ftl
Faculty Member (2)
Librarian (1)
Person (3)
Faculty Member
all A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Baker, Able
menupage-browse.ftl
Faculty, Jane
Assistant Professor
page.ftl
footer.ftl
©2014 VIVO Project | Terms of Use | Powered by VIVO | Version rel-1.7-rc2-38-gbb7ab3b tag
About
Support

The following templates are used in the People page, and in other pages that allow users to browse through a class group.

```
pageSetup.ftl
page.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    search.ftl
    menu.ftl
        developer.ftl
    page-classgroup.ftl
        menupage-checkForData.ftl
        menupage-browse.ftl
        menupage-scripts.ftl
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	Purpose	From

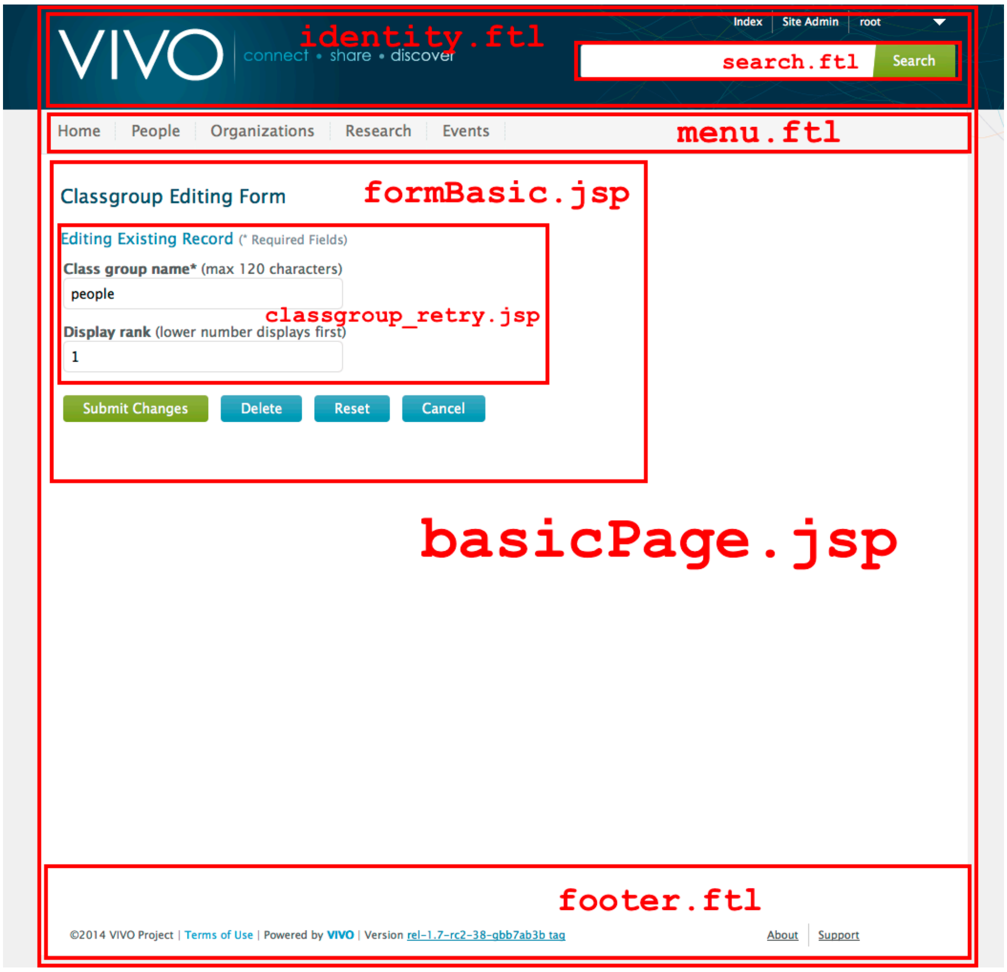
`pageSetup.ftl`
`page.ftl`
`head.ftl`
`stylesheets.ftl`
`headScripts.ftl`
`identity.ftl`
 	
`languageSelector.ftl`
`search.ftl`
`menu.ftl`
`developer.ftl`
	as above.	

`page-classgroup.ftl`	Combines the components to create an AJAX-driven page that browses among the classes in a class group.	VIVO is configured to use this template in the `People` menu page page. **You can change this configuration: see** Menu and page management (see page 163)**.** This template is invoked by `ClassGroupPageData.java`, which is assigned to the `People` page in `menu.n3`.
`menupage-checkForData.ftl`	Checks to see if the page will be empty. Displays messages suitable to a VIVO administrator or to another user, depending on who is viewing the page.	Included by `page-classgroup.ftl`.
`menupage-browse.ftl`	Creates the page context that will be filled by AJAX calls.	Included by `page-classgroup.ftl`.
`menupage-scripts.ftl`	Creates or links to the JavaScripts used in browsing among classes of individuals.	Included by `page-classgroup.ftl`.

`footer.ftl`
`scripts.ftl`
`googleAnalytics.ftl`
	as above.	

8.3.17.4 A back-end page

VIVO provides several pages that allow administrators to edit the classes and properties in the ontology, and to create or adjust class groups and property groups. These pages are built around the older JSP (Java Server Pages) technology, although the header and footer are created from the same Freemarker templates as other pages.


VIVO
connect • share • discover
identity.ftl
Index | Site Admin | root
search.ftl
Search
Home | People | Organizations | Research | Events
menu.ftl
formBasic.jsp
classgroup_retry.jsp
Submit Changes
Delete
Reset
Cancel
basicPage.jsp
footer.ftl
About | Support

Classgroup Editing Form

Editing Existing Record (* Required Fields)

Class group name* (max 120 characters)
people

Display rank (lower number displays first)
1

©2014 VIVO Project | Terms of Use | Powered by VIVO | Version rel-1.7-rc2-38-gbb7ab3b tag

The following templates and JSPs are used in creating this page.

```
basicPage.jsp
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    search.ftl
    menu.ftl
        developer.ftl
    formBasic.jsp
        classgroup_retry.jsp
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	Purpose	From
`basicPage.jsp`	The master template for the VIVO back-end pages.	Specified by `ClassgroupRetryController.java`.

`head.ftl`
`stylesheets.ftl`
`headScripts.ftl`
`identity.ftl`
 	
`languageSelector.ftl`
`search.ftl`
`menu.ftl`
`developer.ftl`
	as above.	

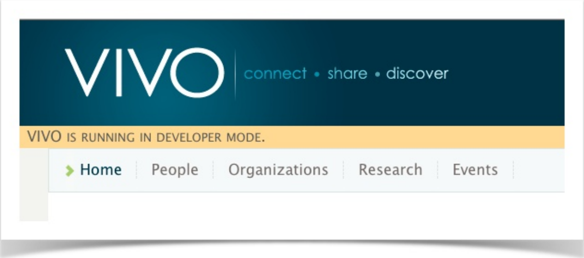
`formBasic.jsp`	A generic frame that provides title and buttons for an edit.	Specified by `ClassgroupRetryController.java`.
`classgroup_retry.jsp`	Shows the fields that may be edited for a class group.	Specified by `ClassgroupRetryController.java`.

`footer.ftl`
`scripts.ftl`
`googleAnalytics.ftl`
	as above.	

8.3.18 Tips for Interface Developers

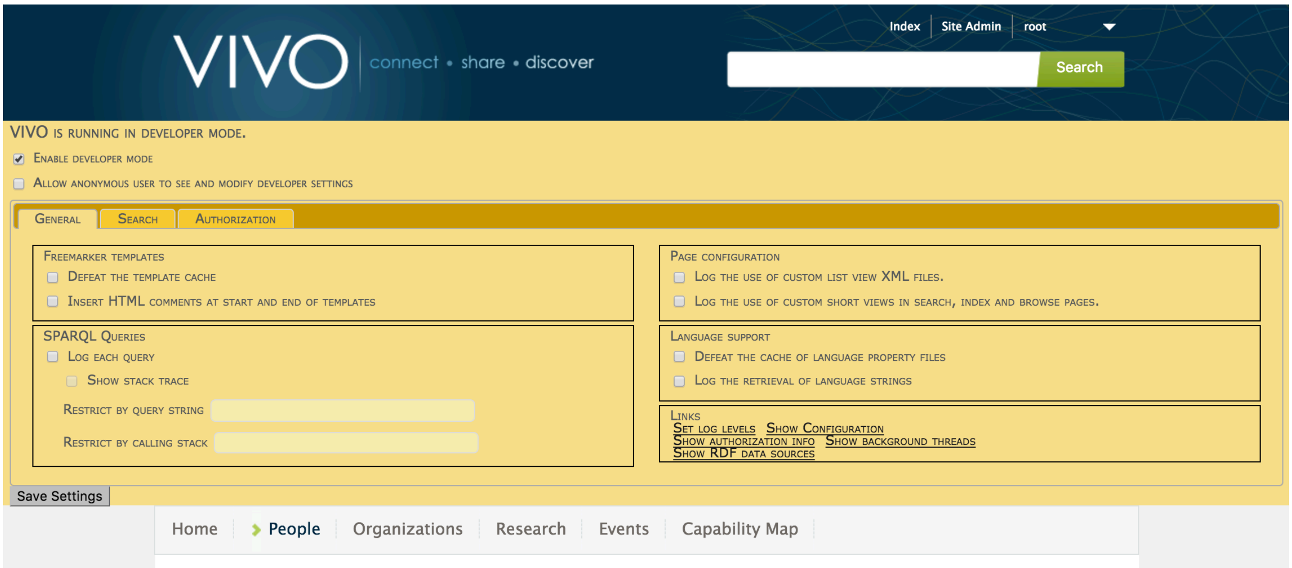
8.3.18.1 Use the Developer Panel

Many of these techniques involve the Developer Panel. You can start the Developer Panel at Site Admin > Activate Developer Panel. When the Developer Panel has been activated, you will see:


VIVO
connect • share • discover
Home
People
Organizations
Research
Events

VIVO is running in developer mode.

When you click on the Developer Mode banner, you will see:


Index
Site Admin
root
VIVO
connect • share • discover
Search
General
Search
Authorization
Save Settings
Home
People
Organizations
Research
Events
Capability Map

VIVO is running in developer mode.

☑ Enable developer mode

☐ Allow anonymous user to see and modify developer settings

Freemarker templates

☐ Defeat the template cache

☐ Insert HTML comments at start and end of templates

SPARQL Queries

☐ Log each query

☐ Show stack trace

Restrict by query string

Restrict by calling stack

Page configuration

☐ Log the use of custom list view XML files.

☐ Log the use of custom short views in search, index and browse pages.

Language support

☐ Defeat the cache of language property files

☐ Log the retrieval of language strings

Links

Set log levels Show Configuration
Show authorization info Show background threads
Show RDF data sources

To close the Developer Panel, unselect "Enable Developer Mode" in the upper left hand corner, and press "Save Settings" in the lower left hand corner.

8.3.18.1.1 Developer Panel Settings

You can change settings on The Developer Panel interactively, while VIVO is running, or you can use a `developer.properties` file in your VIVO home directory.

A typical developer.properties file

```
developer.enabled=true
developer.permitAnonymousControl=true
developer.defeatFreemarkerCache=true
```

When any feature of The Developer Panel is active, you will see this indicator in the header of your VIVO pages:

This is to remind you that developer options may slow down your VIVO, and should not be used in production.

8.3.18.2 Iterate your code more quickly

8.3.18.2.1 Don't restart VIVO until you need to

VIVO will detect changes to the templates without requiring a restart. However, you will probably want to defeat the Freemarker cache (see below).

Also, VIVO will serve the latest version of CSS, JavaScript, or image files. For these files, however, you may need to clear the cache in your browser. Instructions for doing this will differ, depending on which browser you are using. If you don't know how to reset the cache in your browser, you may want to consult this web site: http://clearyourcache.com/, or just search the web for "clear browser cache".

If you change any other types of files, you will need to restart VIVO after running the build script.

8.3.18.2.2 Defeat the Freemarker cache

As mentioned above, VIVO will detect changes to Freemarker templates. By default, however, VIVO will not detect the changes immediately. The Freemarker framework caches the templates that it uses, and won't even look to see if a template has changed until 1 minute after it was last read from disk. In a production system, of course, that makes the accessing much more efficient. When you are making frequent changes, it's an annoyance.

Use The Developer Panel to defeat the Freemarker cache.

8.3.18.2.3 Customizing listViewConfigs

Ted Lawless has written an open-source Python script to assist with viewing the output of a listViewConfig[122] without having to rebuild the entire Vivo app.

Also, you can skip the unit tests when building VIVO. Unit tests do not apply to listViewConfigs.

8.3.18.3 Reveal what VIVO is doing

8.3.18.3.1 Insert template delimiters in the HTML

It's not always clear which template has created a particular piece of your HTML page. Templates include other templates, templates are invoked in custom list views, short views, etc. You can use The Developer Panel to insert comments in the HTML that tell you where each template begins and ends.

For example, this section of a page was produced mostly by the `identity.ftl` template. The `languageSelector.ftl` template is included, but does not generate any HTML. The next section is produced by the `menu.ftl` template, and so on.

```
...
```

---

[122] http://lawlesst.github.io/notebook/vivo-listview.html

```
    <body >
            <!-- FM_BEGIN identity.ftl -->

<header id="branding" role="banner">

  <h1 class="vivo-logo"><a title="VIVO | connect share discover" href="/vivo">
    <span class="displace">VIVO</span>
  </a></h1>

  <nav role="navigation">
    <ul id="header-nav" role="list">
<!-- FM_BEGIN languageSelector.ftl -->
<!-- FM_END languageSelector.ftl -->
        <li role="listitem"><a href="/vivo/browse" title="Index">Index</a></li>
        <li role="listitem"><a href="/vivo/siteAdmin" title="Site Admin">Site Admin</
a></li>
        <li>
          <ul class="dropdown">
            <li id="user-menu"><a href="#" title="user">Jim</a>
              <ul class="sub_menu">
                <li role="listitem"><a href="/vivo/accounts/myAccount" title="My
account">My account</a></li>
                <li role="listitem"><a href="/vivo/logout" title="Log out">Log out</
a></li>
              </ul>
            </li>
          </ul>
        </li>
      </ul>
    </nav><!-- FM_END identity.ftl -->
            <!-- FM_BEGIN menu.ftl -->
</header>

...
```

8.4 Adding Additional Ontologies to VIVO

8.4.1 Overview

VIVO provides the capability to add ontologies and use them to describe your data and for query. You can add your own, or pre-existing ontologies. In each case, the ontology to be added will appear in the VIVO Ontology List along with ontologies provided with VIVO.

If your ontology assertions contain an owl:Ontology assertion giving the URI of your ontology, your ontology will appear in Site Admin → Ontology List. Otherwise, your ontology assertions will be added to be VIVO and will be fully functional as classes, object properties, and data properties, but will not be associated with a named ontology.

Ontologies added to VIVO should use the .owl file type, and should be written in OWL DL. Ontologies written in RDF or OWL FULL are not supported by VIVO.

8.4.2 Using the Web Interface

8.4.2.1 Create an Ontology from the Web Interface

To create an ontology from the web interface, follow the steps in Create, Assign, and Use an Institutional Internal Class (see page 65). These instructions show how to create an ontology using the web interface. Once your ontology is created, you can add an Institutional Class as described, additional classes, data properties and Object properties. You may create as many additional ontologies as you need.

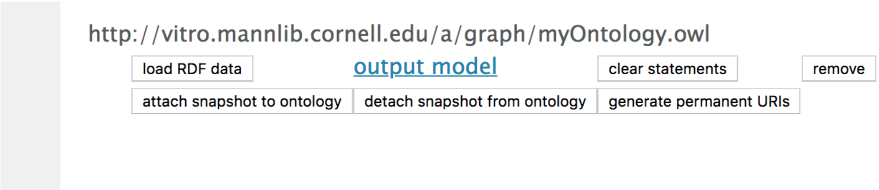
8.4.2.2 Load an Existing Ontology as Data

You can add an ontology from a file into VIVO using the Add/Remove RDF Data feature shown on the Site Admin page. This method is simple, but not recommended for production sites. Your ontology will be loaded into the VIVO main data graph and could be very difficult to remove. Much preferred for production sites is the following method, Loading to a Named Graph.

8.4.2.3 Load an Existing Ontology to a Named Graph

Use Site Admin → Ingest Tools → Manage Jena Models → Create Jena Model

Give your named graph (Jena calls them models) a name such as myOntology.owl



http://vitro.mannlib.cornell.edu/a/graph/myOntology.owl

You will see your ontology represented as an empty named graph in the list of Graphs.

Click Load RDF data, and upload your ontology file to your named graph.

In this way, you keep your ontology assertions separate from other VIVO assertions. You can drop the graph, and/or reload it as needed.

8.4.3 Loading from Filegraphs

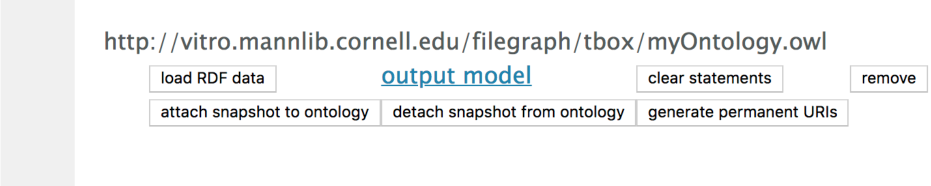
There is another mechanism for incorporating ontologies into VIVO. This involves "filegraphs," and is how the VIVO ontology is included with the software. Filegraphs are RDF documents stored in the VIVO home directory. Each filegraph corresponds to a single named graph in the triple store. Every time Tomcat starts, VIVO checks each of these graphs to ensure that its contents exactly match the triples found in the corresponding file. If the file has changed, VIVO makes the necessary modifications to the corresponding named graph in the triple store. If a filegraph is removed from its directory, its graph in the triple store will be deleted the next time Tomcat starts.

8.4.3.1 Example

```
vivo.home/
    rdf/
        tbox/
            filegraph/
                vivo.owl
                ontologies.owl
                ...
                myOntology.owl
                ...
```

Adding myOntology.owl to the directory as shown above will automatically create the corresponding named graph in the triple store after Tomcat is restarted:

After restart, go to Site Admin → Ingest Data → Manage Jena Models. Review the named graphs. You should see:



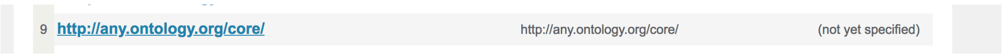
http://vitro.mannlib.cornell.edu/filegraph/tbox/myOntology.owl

load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs

Removing the myOntology.owl file from the filegraph directory and restarting Tomcat will automatically remove the named graph. This will remove all ontology assertions made in myOntology.owl. It will not remove any abox assertions using myOntology.owl from the content triple store.

8.4.4 Namespace Prefixes

owl:Ontology statements result in the ontology being added to VIVO's Ontology List. You might see an entry such as:



9 http://any.ontology.org/core/ http://any.ontology.org/core/ (not yet specified)

Indicating that the ontology you have added does not have a label or prefix.

Specify a label and a prefix for your ontology as shown below:

```
    <owl:Ontology rdf:about="http://any.ontology.org/core/">
        <rdfs:label xml:lang="en-US">My Ontology</rdfs:label>
        <vitro:ontologyPrefixAnnot xml:lang="en-US">myo</vitro:ontologyPrefixAnnot>
    </owl:Ontology>
```

Restarting Tomcat to reload the filegraph will display your prefix and label.

8.5 Enable an external authentication system

- How User Accounts are Associated with Profile Pages (see page 265)
- Shibboleth example (see page 266)
- Using a Tomcat Realm for external authentication (see page 269)

8.5.1 How User Accounts are Associated with Profile Pages

There are three elements in the linkage between a User Account and a Profile page:

- The user account holds the `externalAuthId`
- `runtime.properties` specifies the `URI` of the matching property
- The profile page must have a property with that `URI` whose value matches the `externalAuthId`. (The property value is either a String or an untyped literal.)

8.5.1.1 A user account may have an externalAuthId

- The `externalAuthId` is optional.
- There are several ways to create a `externalAuthId`:
  - If you are using internal authentication – managed within VIVO — then each account must be created by an admin, and the admin may choose to set the externalAuthId to a useful value.
  - If you are using external authentication – Shibboleth, or the like – then when a user without an account passes authentication, an account is created auto-magically. The externalAuthId is set to the user's Shibboleth ID.
  - Regardless of the type of authentication, you could choose to ingest the information for the user accounts, and create the externalAuthId as part of that ingest.
- In any case, the externalAuthId can be used to link to the user's profile page.
- This info is stored in the userAccounts model.
- You can confirm this by going to the SiteAdmin page, clicking on "Ingest Tools", then "Manage Jena Models", then the button labelled "RDB Models", then the "output model" link under vitro-kb-userAccounts. The output should contain statements that look something like this:

```
<http://vivo.mydomain.edu/individual/u8041>
a <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#UserAccount> ;
<http://vitro.mannlib.cornell.edu/ns/vitro/authorization#emailAddress>
"jeb228@cornell.edu"^^xsd:string ;
<http://vitro.mannlib.cornell.edu/ns/vitro/authorization#externalAuthId>
"jeb228"^^xsd:string ;
```

- I don't know what would happen to a user with more than one one externalAuthID. Probably VIVO will arbitrarily choose among them.

8.5.1.2 runtime.properties may contain a value for selfEditing.idMatchingProperty

- You can confirm this value by looking in the `vivo.all.log` file in Tomcat logs. Each time VIVO starts up, the first entry written to the log contains all of the properties from `runtime.properties`. It helps to inspect this if you might possibly be reading the wrong runtime.properties file.
- At Cornell, ours looks like this:

```
   selfEditing.idMatchingProperty = http://vivo.cornell.edu/ns/hr/0.9/
hr.owl#netId
```

8.5.1.3 The profile page may match the externalAuthId on the user account

- To associate a profile page with a user account, the Individual must have a data property whose URI is the one from runtime.properties, and whose value is equal to the externalAuthId of the user account.
- For example, the Individual object that forms the basis for my profile page contains a statement like this:

```
<http://vivo.cornell.edu/individual/JamesBlake>
    <http://vivo.cornell.edu/ns/hr/0.9/hr.owl#netId>
    "jeb228"
    .
```

- You can confirm this by logging in as an admin, navigating to the profile page, clicking on "edit this individual" and then the button labelled "Raw Statements with this Resource as Subject"
- In the example above, the "netId" field is set to an untyped Literal. A String Literal will work also.

8.5.2 Shibboleth example

Almost all of the configuration described on this page is not unique to VIVO and will likely vary by your own institution's Shibboleth configuration. That said, an example configuration is provided to hopefully help implementers move their own setup in the right direction.

8.5.2.1 Install the Shibboleth module for Apache

Installation of the Shibboleth service provider module will depend on your OS distribution. Some documentation on installation is available here: https://shibboleth.atlassian.net/wiki/spaces/SP3/pages/2065335547/LinuxInstall.

8.5.2.2 Edit Shibboleth Application Defaults

Edit defaults. Example location is /etc/shibboleth/shibboleth2.xml. Add in your site's URL as the entityID, the attribute name Shibboleth will provide the user ID in, and set attributePrefix so Apache's AJP will pass the attributes through. The attribute name in this example is 'eppn' but may be different depending on your Shibboleth configuration.

shibboleth2.xml

```
    <!-- The ApplicationDefaults element is where most of Shibboleth's SAML bits are defined. -->
    <ApplicationDefaults entityID="https://vivo.school.edu/shibboleth" REMOTE_USER="eppn" attributePrefix="AJP_">
```

8.5.2.3 Secure VIVO's 'special page' at /loginExternalAuthReturn

Somewhere in your Apache configuration, load the Shibboleth module and secure the pages.

httpd.conf

```
LoadModule mod_shib /usr/lib64/shibboleth/mod_shib_24.so

<Location /Shibboleth.sso>
  AuthType None
  Require all granted
</Location>

<Location /loginExternalAuthReturn>
  AuthType shibboleth
  ShibRequestSetting requireSession 1
  require shib-session
</Location>
```

8.5.2.4 Allow Shibboleth's pages to be served by Apache

Exempt Shibboleth's pages from the Tomcat/VIVO proxy. Example Virtualhost:

ssl.conf

```
<VirtualHost _default_:443>
  ServerName vivo.school.edu
  ProxyPass /Shibboleth.sso/* !
  ProxyPassMatch "/Shibboleth.sso/.*/" !
  ProxyPass / ajp://localhost:8009/ retry=15 secret=your_tomcat_secret timeout=600
</VirtualHost>
```

8.5.2.5 Allow Shibboleth attributes to be passed through to Tomcat

As a security feature, Tomcat does not pass through request attributes to applications unless they meet a specific pattern. You can specify the allowed attributes in regex by adding 'allowedRequestAttributesPattern' to your AJP connector definition.

Example AJP connector config in Tomcat's server.xml.

server.xml

```
    <!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector protocol="AJP/1.3"
               address="::1"
               port="8009"
               redirectPort="8443"
               secretRequired="true"
               secret="your_tomcat_secret"
               URIEncoding="UTF-8"
               tomcatAuthentication="false"
               allowedRequestAttributesPattern="^(Shib-.*|eppn)$"
               />
```

8.5.2.6 Edit runtime.properties

Add the header or attribute name Shibboleth will use to provide VIVO the user's ID.

runtime.properties

```
externalAuth.netIdHeaderName = eppn
```

Specifying externalAuth.netIdHeaderName will activate the external authentication in VIVO. Restart Shibboleth, Apache, Tomcat, and VIVO to allow your changes to take effect.

8.5.3 Using a Tomcat Realm for external authentication

8.5.3.1 Background

VIVO is not written to use the standard JEE or Tomcat authentication systems, so using a Tomcat Realm would require some customization. This doesn't seem very difficult, it just hasn't been a priority for us.

When VIVO is set up to use external authentication, it uses a reverse-proxy setup, where an Apache HTTP server intercepts all calls to Tomcat. The Apache server uses a Shibboleth module or other module to secure a particular page: http://localhost:8080/vivo/loginExternalAuthReturn.

If an HTTP request is made to that page, and the request does not belong to a session that is already logged it, the Shibboleth module in Apache will intercept the request and guide the user through the authentication process. When the user's credentials are accepted, the module invokes the secured page, as requested, storing the user's ID in one of the HTTP headers. The VIVO code reads the user ID from the HTTP header and stores it in the session object. Only that one page is secured, and VIVO remembers the user ID for use in subsequent requests.

Which HTTP header will VIVO inspect for the user ID? The header which is named in externalAuth.netIdHeaderName.

Most institutions that use VIVO also use Shibboleth in their web applications, or something with a similar mechanism. The IT group at the institution provides the VIVO implementers with the appropriate Apache module and configuration information.

I don't know of anyone who has tried to use a Tomcat Realm to accomplish external authentication in VIVO. I think it would require some small modification of the VIVO code, perhaps a change to ExternalAuthHelper.getExternalAuthId(). Tomcat would use the Realm to create a Principal object in the HTTP request, and VIVO would get the user ID from that Principal instead of looking in an HTTP header. Web.xml would be modified to secure the page, as you have already done.

8.5.3.2 Testing

It really was just that easy!

I added these lines to ExternalAuthHelper.getExternalAuthId(), right after the check for a null request object:

```
Principal p = request.getUserPrincipal();
if (p != null) {
   log.debug("Found a UserPrincipal in the request: " + p);
   String userId = p.getName();
   if (StringUtils.isNotEmpty(userId)) {
       log.debug("Got external auth from UserPrincipal: " + userId);
       return userId;
   }
}
```

I added these lines to the end of web.xml, just before the closing </web-app>:

```
<security-constraint>
```

```
    <web-resource-collection>
        <web-resource-name>ExternalAuthPage</web-resource-name>
        <url-pattern>/loginExternalAuthReturn</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>tomcat</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
```

I set this property in deploy.properties:

```
externalAuth.buttonText = Log in using basic Tomcat
```

And voila, my tomcat-users.xml file is my external authentication system!

Obviously, you will want to use FORM authentication, instead of BASIC, and something other than the default Realm. But I expect you know how to do that already.

Please, let me know how this progresses for you. This may be something that we will add to the next release.

Jim Blake

8.6 Authorization

- Writing a controller for a secured page (see page 270)
- Creating a VIVO authorization policy - an example (see page 275)
- A more elaborate authorization policy (see page 283)
- The IdentifierBundle - who is requesting authorization? (see page 289)

8.6.1 Writing a controller for a secured page

8.6.1.1 Concepts

8.6.1.1.1 A secured page

A secured page in VIVO is one that can not be viewed by the general public. If an unauthorized user attempts to view a secured page, even by entering the URL directly into a browser, the attempt should fail.

8.6.1.1.2 How is a page secured?

To secure a page, the controller code requests authorization to perform a particular `RequestedAction`. If the user is not authorized to perform that action, the controller rejects the request. For example, the `RevisionInfoController` checks to see whether the user is authorized for the `SEE_REVISION_INFO.ACTION`. If the user is not authorized for that action, they will not see the Revision Info page.

Other controllers use more complicated tests to determine whether a user is authorized. For example, the `ManageProxiesAjaxController` permits access by any user who is authorized for either the `MANAGE_PROXIES.ACTION` or the `MANAGE_OWN_PROXIES.ACTION`.

8.6.1.1.3 Who may view a secured page?

A secured page can never be viewed by someone who is not logged in to VIVO. Since we don't know who the user is, we can't know whether they are authorized to view the page.

If a user is logged in, there is a list of `Identifier`s associated with their account. The `Identifier`s are pieces of information about that user, including their account URI, the URI of their profile page, their assigned role, any proxy permissions, and more. When a secured page is requested, these `Identifier`s are passed to the list of active `Policy` objects. Each `Policy` applies its own logic to determine whether the user may view the secured page.

8.6.1.1.4 What happens if the user is not authorized?

- If the user is logged in, but does not have authorization to view the secured page, the browser will be redirected to the VIVO home page. A message at the top of the page will state that the user is not authorized to view the page he requested.
- If the user is not logged in, the browser will be redirected to the VIVO login page. When the user logs in, the browser will be redirected to the secured page, and the test is repeated.
    - If the user is authorized, the secured page will be displayed.
    - If the user is not authorized, the home page will be displayed, as previously described.

8.6.1.1.5 What happens when a user logs out?

If a user is viewing an unsecured page, and clicks on the "Log out" link, the page will be refreshed. For some pages, particularly profile pages, the contents of the page may have changed. Many people appreciate this feature when editing their own profiles. Log in, and you can edit. Log out, and you can see what your page looks like to the public.

If a user is viewing a secured page and clicks on the "Log out" link, the browser will be redirected to the VIVO home page.

8.6.1.2 Requested Actions

Requested actions are usually quite simple. For example, the `RevisionInfoController` requests permission to display the revision info page. The user either has that permission or they do not.

On the other hand, Requested actions can be quite detailed. For example, the `ImageUploadController` requests permission to add or modify a particular triple in the data model. If the user is logged in as root or admin, they have permission. However, if the user is logged in as a self-editor, a complex algorithm is performed to determine whether they are authorized to add or modify the triple in question. They may be authorized because the subject of the triple is the URI of their own profile page, or because they have been given proxy rights to edit the page in question, or several other possibilities.

8.6.1.3 The most common case

The most common scenario for a secured page is when a simple, unparameterized action is requested, and the user either

- has a permission that provides authorization, or
- does not have that permission and is not authorized.

8.6.1.3.1 The steps

8.6.1.3.1.1 Decide on a permission and requested action

Simple permissions like this are usually implemented by the `SimplePermission` class, which also provides an implementation for the corresponding `RequestedAction`.

In some cases, it makes sense to re-use an existing instance of `SimplePermission`. So for example, `SimplePermission.USE_ADVANCED_DATA_TOOLS_PAGES` authorizes the user for any and all of the RDF ingest/export pages. In other cases, it makes more sense to create a new instance. So `SimplePermission.MANAGE_PROXIES` stands alone with only one usage.

For this example, we will look at `SimplePermission.SEE_REVISION_INFO`, which has only one usage.

8.6.1.3.1.2 Write the controller to require the requested action

If the controller extends `FreemarkerHttpServlet`, override the `requiredActions()` method, like this:

```
@Override
protected Actions requiredActions(VitroRequest vreq) {
    return SimplePermission.SEE_REVISION_INFO.ACTIONS;
}
```

If the controller exends `VitroHttpServlet` (but not `FreemarkerHttpServlet`), add a test to the `doGet()` and `doPost()` methods, like this:

```
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
    if (!isAuthorizedToDisplayPage(req, resp,
            SimplePermission.SEE_REVISION_INFO.ACTIONS)) {
        return;
    }
...
```

Both of these examples take advantage of the fact that each instance of `SimplePermission` defines its own `RequestedAction`, as well as its own `Actions` set.

8.6.1.3.1.3 Grant the permission to the desired users.

Each `Permission`, simple or otherwise, can be assigned to `PermissionSet`s within VIVO. Each user account is associated with a `PermissionSet`, and may use the `Permission`s associated with it. The assignment of `Permission`s to `PermissionSet`s occurs in the file called

```
[vitro]/webapp/rdf/auth/everytime/permission_config.n3
```

By inspecting the RDF in this file, we can see that the `SEE_REVISION_INFO` permission is assigned to `ADMIN`, `CURATOR`, and `EDITOR` `PermissionSet`s. Here is an excerpt of the file with the relevent RDF:

```
@prefix auth: <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#> .
@prefix simplePermission:
<java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#> .

auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
auth:CURATOR auth:hasPermission simplePermission:SeeRevisionInfo .
auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
```

In future versions of VIVO, the `Permission/PermissionSet` framework may be extended to permit multiple `PermissionSet`s per user, with GUI-based configuration.

8.6.1.4 A more complex example

TBD

> - It's all about the action that your controller is requesting, and whether your user has authorization to do it.
>   - Actions can be parameterized (modify this statement) or not (see the revision info page)
>   - Authorization can come from a policy, or from a permission
>   - Permissions can be simple, or as complex as a policy
> - Look at the simplest case: RevisionInfoController
>   - Not parameterized: SimplePermission.something.ACTION
> - Code in HttpServlet, FreemarkerServlet, JSP
> - Look at a complex case: ImageUploadController
>   - Also ManageProxiesAjaxController
> - In some cases, it isn't a question of whether your controller will run, but what it will do:
>   - BaseIndividualTemplateModel
>   - ```
>      public boolean isEditable() {
>         AddDataPropertyStatement adps = new AddDataPropertyStatement(
>             vreq.getJenaOntModel(), individual.getURI(),
>             RequestActionConstants.SOME_URI);
>         AddObjectPropertyStatement aops = new AddObjectPropertyStatement(
>             vreq.getJenaOntModel(), individual.getURI(),
>             RequestActionConstants.SOME_URI,
>             RequestActionConstants.SOME_URI);
>         return PolicyHelper.isAuthorizedForActions(vreq, new
>     Actions(adps).or(aops));
>      }
>     ```
>   - LoginRedirector
>     - ```
>        private boolean canSeeSiteAdminPage() {
>           return PolicyHelper.isAuthorizedForActions(request,
>               SimplePermission.SEE_SITE_ADMIN_PAGE.ACTIONS);
>        }
>       ```
>   - BaseSiteAdminController

```
•       if (PolicyHelper.isAuthorizedForActions(vreq,
    SimplePermission.MANAGE_USER_ACCOUNTS.ACTIONS)) {
            data.put("userAccounts", UrlBuilder.getUrl("/accountsAdmin"));
        }
```

8.6.2 Creating a VIVO authorization policy - an example

8.6.2.1 Overview

The ability of users to access data in VIVO is controlled by a collection of Policy objects. By creating or controlling Policy objects, you can control access to the data.

The Policy objects are instances of Java classes that implement the `PolicyIface` interface. These objects are created when VIVO starts up, and are collected in the `ServletPolicyList`. When code in VIVO needs to know whether a user is authorized to perform a particular action, the code creates a `RequestedAction` object and passes it to the Policy list for approval.

When the list is asked for approval, the first Policy in the list is asked first. It must respond with a decision that is `AUTHORIZED`, `UNAUTHORIZED`, or `INCONCLUSIVE`. If the decision is `AUTHORIZED` or `UNAUTHORIZED`, it is taken to be final, and the other Policies in the list are not consulted. If the decision is `INCONCLUSIVE`, then the next Policy in the list is asked to approve the same request, and the process repeats until a conclusive answer is obtained, or until all policies have answered. If no Policy has answered with `AUTHORIZED`, the request fails.

The code below is an example of such a Policy. The entire class is available in the attached file[123].

8.6.2.2 The example

This Policy will check each request to edit an object property statement. The request will be rejected if the statement appears in any graph that is not in the approved set.

The use case is where an individual whose data is stored in the default graph `(vitro-kb2`) links to data in other graphs which were created by ingest and may not be edited. The result of this Policy is that there will be no edit link from the profile page of the individual to that data.

---

[123] https://wiki.lyrasis.org/download/attachments/260899130/RestrictEditingByGraphPolicy.java?api=v2&modificationDate=1664883363133&version=1

8.6.2.2.1 Lines 1-39: imports

```
1   /* $This file is distributed under the terms of the license in /doc/license.txt$ */
2
3  package edu.cornell.mannlib.vitro.webapp.auth.policy;
4
5  import java.util.ArrayList;
6  import java.util.Arrays;
7  import java.util.Collections;
8  import java.util.HashSet;
9  import java.util.List;
10 import java.util.Set;
11
12 import javax.servlet.ServletContext;
13 import javax.servlet.ServletContextEvent;
14 import javax.servlet.ServletContextListener;
15
16 import org.apache.commons.logging.Log;
17 import org.apache.commons.logging.LogFactory;
18
19 import com.hp.hpl.jena.query.Dataset;
20 import com.hp.hpl.jena.query.Query;
21 import com.hp.hpl.jena.query.QueryExecution;
22 import com.hp.hpl.jena.query.QueryExecutionFactory;
23 import com.hp.hpl.jena.query.QueryFactory;
24 import com.hp.hpl.jena.query.ResultSet;
25 import com.hp.hpl.jena.query.Syntax;
26 import com.hp.hpl.jena.rdf.model.RDFNode;
27 import com.hp.hpl.jena.shared.Lock;
28
29 import edu.cornell.mannlib.vitro.webapp.auth.identifier.IdentifierBundle;
30 import edu.cornell.mannlib.vitro.webapp.auth.identifier.common.IsRootUser;
31 import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.Authorization;
32 import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyDecision;
33 import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyIface;
34 import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
35 import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.propstmt.EditObjectPropertyStatement;
36 import edu.cornell.mannlib.vitro.webapp.dao.jena.QueryUtils;
37 import edu.cornell.mannlib.vitro.webapp.servlet.setup.JenaDataSourceSetupBase;
38 import edu.cornell.mannlib.vitro.webapp.startup.StartupStatus;
```

Import statements for the classes used in the Policy

8.6.2.2.2 Lines 40-56: Class declaration, variables, constructor

```
 /**
 * Deny authorization to edit a statement from one of the prohibited graphs.
 */
public class RestrictEditingByGraphPolicy implements PolicyIface {
    private static final Log log = LogFactory
            .getLog(RestrictEditingByGraphPolicy.class);

    private static final Syntax SYNTAX = Syntax.syntaxARQ;
    private static final Set<String> PERMITTED_GRAPHS = new HashSet<>(
            Arrays.asList(new String[] { "http://vitro.mannlib.cornell.edu/default/vitro-kb-2" }));

    private final Dataset dataset;

    public RestrictEditingByGraphPolicy(ServletContext ctx) {
        this.dataset = JenaDataSourceSetupBase.getStartupDataset(ctx);
    }
```

The class must implement the `PolicyIface` interface.

The constructor stores a reference to the `startupDataset`, which will be used to execute SPARQL queries. Because this reference is taken from the context, it will contend with all other context-based references for access to a single database connection. It would be more efficient to use a `Dataset` that was provided by the `HttpServletRequest`, but a Policy never has access to the Request. This will be changed in a future release. (See this JIRA issue[124].)

The `PERMITTED_GRAPHS` constant holds the set of graph URIs for which editing is permitted. It would be a simple code change to use a `PROHIBITED_GRAPHS` constant instead.

8.6.2.2.3 Lines 57-68: Implement the isAuthorized() method

```
    /**
     * For each request to Edit an ObjectProperty, find out what graph the
     * statement is in. Prohibit editing if the statement is in the wrong graph.
     *
     * Note that this will not work with a DataProperty, since the
     * EditDataProperty object does not contains the value of the property. We
     * didn't anticipate that editing privileges would be determined by the
```

---

[124] https://jira.duraspace.org/browse/VIVO-269

```
     * contents of the string.
     */
    @Override
    public PolicyDecision isAuthorized(IdentifierBundle whoToAuth,
            RequestedAction whatToAuth) {
```

Every `PolicyIFace` class must implement this method.

- `whoToAuth` is a collection of `Identifier`s, each one holding a piece of information about the user who is currently logged in.
- `whatToAuth` is the action being requested.

8.6.2.2.4 Lines 69-81: Make quick and easy decisions

```
        if (whoToAuth == null) {
            return inconclusiveDecision("whoToAuth was null");
        }
        if (whatToAuth == null) {
            return inconclusiveDecision("whatToAuth was null");
        }
        if (IsRootUser.isRootUser(whoToAuth)) {
            return inconclusiveDecision("Anything for the root user");
        }
        if (!(whatToAuth instanceof EditObjectPropertyStatement)) {
            return inconclusiveDecision("Only interested in editing object properties");
        }
```

Policies are called very frequently, especially when a large profile page is displayed. Whenever possible, answer the easy questions first before doing more expensive tests.

Checking for `null` arguments should not be necessary - these arguments should never be null. However, it is simple defensive programming, and not costly.

This policy is only interested in requests to edit object property statements, so we can quickly reject any other type of `RequestedAction`. Again, the `INCONCLUSIVE` decision is equivalent to saying "let someone else decide."

This policy does not attempt to restrict the editing of data property statements. This is because the `EditDataPropertyStatement` class does not include the value of the data property. At one time it was felt that this could not affect the decision of whether to permit the request. This will be changed in a future release (See this JIRA issue[125]).

This policy will not restrict the root account from attempting to edit statements.

---

[125] https://jira.duraspace.org/browse/VIVO-268

We already have `RootUserPolicy`, which says that the root user is permitted to do anything. So why do we need this test?
We need to consider the order in which policies are called, and to remember that polling ono a `RequestedAction` will stop when any policy returns a decision that is not `INCONCLUSIVE`. So, if this Policy is placed before `RootUserPolicy`, and returns an `UNAUTHORIZED` decision, then the `RootUserPolicy` will never been consulted.
The question of "what to do when one Policy would authorize and another Policy would prohibit" is a tricky one.

8.6.2.2.5 Lines 82-105: Execute the SPARQL query and test the result

```
        EditObjectPropertyStatement stmt = (EditObjectPropertyStatement) whatToAuth;

        String queryString = assembleQueryString(stmt);
        List<String> graphUris = executeQuery(queryString);
        log.debug("graph URIs: " + graphUris);

        if (graphUris.isEmpty()) {
            log.warn("Can't find this statement in any graph: " + stmt);
            return inconclusiveDecision("Can't find this statement in any graph: "
                    + stmt);
        }

        graphUris.removeAll(PERMITTED_GRAPHS);
        if (graphUris.isEmpty()) {
            log.debug("Permitted: " + stmt);
            return inconclusiveDecision("Statement is only in permitted graphs: "
                    + stmt);
        }

        log.debug("Statement is prohibited: " + stmt + ", graphs=" + graphUris);
        return unauthorizedDecision("Statement is in a prohibited graph, "
                + stmt + " in " + graphUris);
    }
```

Assemble the query and execute it. This results in a list of the URIs of all Graphs that contain this statment. (See the subroutines in the next section).

What to do if we do not find the statement in any graph? It would be possible to err on the side of caution and return an `UNAUTHORIZED` decision. We could even throw a `RuntimeException` of some sort to abort the page display. In this case, we choose to return `INCONCLUSIVE` and write a warning to the log.

If the statement appears only in the permitted graphs, return a decision of `INCONCLUSIVE`, letting some other policy decide.

If the statement appears in other, prohibited graphs, return a decision of `UNAUTHORIZED`, rejecting the requested action.

8.6.2.2.6 Lines 106-171: Subroutines

```
    private static final String QUERY_TEMPLATE = "" + //
            "SELECT ?graph WHERE{" + //
            "   GRAPH ?graph{" + //
            "     ?s ?p ?o ." + //
            "   } " + //
            "} LIMIT 10"; //

    private String assembleQueryString(EditObjectPropertyStatement stmt) {
        String q = QUERY_TEMPLATE;
        q = QueryUtils.subUriForQueryVar(q, "s", stmt.getSubjectUri());
        q = QueryUtils.subUriForQueryVar(q, "p", stmt.getPredicateUri());
        q = QueryUtils.subUriForQueryVar(q, "o", stmt.getObjectUri());
        return q;
    }
```

We have a template for the SPARQL query. Substitute the values for this statement into the query. The only unresolved variable will be `?graph`.

```
    private List<String> executeQuery(String queryStr) {
        log.debug("select query is: '" + queryStr + "'");
        QueryExecution qe = null;
        dataset.getLock().enterCriticalSection(Lock.READ);
        try {
            Query query = QueryFactory.create(queryStr, SYNTAX);
            qe = QueryExecutionFactory.create(query, dataset);
            return parseResults(queryStr, qe.execSelect());
        } catch (Exception e) {
            log.error("Failed to execute the Select query: " + queryStr, e);
            return Collections.emptyList();
        } finally {
            if (qe != null) {
                qe.close();
            }
            dataset.getLock().leaveCriticalSection();
        }
```

```java
    }

    private List<String> parseResults(String queryStr, ResultSet results) {
        List<String> uris = new ArrayList<>();
        if (results.hasNext()) {
            try {
                RDFNode node = results.next().get("graph");
                if ((node != null) && node.isResource()) {
                    uris.add(node.asResource().getURI());
                }
            } catch (Exception e) {
                log.warn("Failed to parse the query result" + queryStr, e);
            }
        }
        return uris;
    }
```

Execute the SPARQL query against the `Dataset`. Extract the graph URIs from the result.

```java
    /**
     * An UNAUTHORIZED decision says
     * "Not allowed. Don't bother asking anyone else".
     */
    private PolicyDecision unauthorizedDecision(String message) {
        return new BasicPolicyDecision(Authorization.UNAUTHORIZED, getClass()
                .getSimpleName() + ": " + message);
    }

    /**
     * An INCONCLUSIVE decision says "Let someone else decide".
     */
    private PolicyDecision inconclusiveDecision(String message) {
        return new BasicPolicyDecision(Authorization.INCONCLUSIVE, getClass()
                .getSimpleName() + ": " + message);
    }
```

Convenience methods for creating `PolicyDecision` return values.

8.6.2.3 Setup when VIVO starts

When VIVO starts execution, the `StartupManager` processes the file `startup_listeners.txt`, and instantiating each class that is named in the file, and invoking the `contextsInitialized()` method on each class.

8.6.2.3.1 Lines 172-193: The Setup class

```java
    // ----------------------------------------------------------------------
    // Setup class - must be specified in startup_listeners.txt before any
    // policy that might be more permissive.
    // ----------------------------------------------------------------------
    public static class Setup implements ServletContextListener {

        @Override
        public void contextInitialized(ServletContextEvent sce) {
            ServletContext ctx = sce.getServletContext();
            StartupStatus ss = StartupStatus.getBean(ctx);

            RestrictEditingByGraphPolicy p = new RestrictEditingByGraphPolicy(
                    ctx);
            ServletPolicyList.addPolicy(ctx, p);
            ss.info(this,
                    "Editing object properties is only permitted in these graphs: "
                            + RestrictEditingByGraphPolicy.PERMITTED_GRAPHS);
        }

        @Override
        public void contextDestroyed(ServletContextEvent sce) { /* nothing */
        }
    }
```

The Setup class must implement `ServletContextListener`.

On startup, create an instance of the Policy, and add it to the `ServletPolicyList`. Produce an informative message for the startup status screen.

On shutdown, there is nothing to be done. If there were resources to be freed or files to be closed, this would be the place to do it.

8.6.2.3.2 Invoking the Setup class

Initialize the policy in startup_listeners.txt

```
edu.cornell.mannlib.vitro.webapp.auth.policy.RestrictEditingByGraphPolicy$Setup
```

Add this line to `startup_listeners.txt` . Consult the note above regarding placement of this Policy relative to the other Policies.

8.6.2.4 A more complicated example

For another example of writing a policy, look at A more elaborate authorization policy (see page 283)

8.6.3 A more elaborate authorization policy

Suppose you want to do something more elaborate than just prohibit access to a page. For example, perhaps you want to have some profiles be accessible only to certain people.

This becomes a more interesting task, because all profiles are presented by the same controller. So how do you tell the controller that a person is authorized to view the page for one profile but not for another?

You must create a `RequestedAction` that takes parameters, and then have your `Policy` use those parameters in its decision.

Another issue is that there are several URLs that will lead to the same profile page. These URLs are equivalent:

Equivalent URLs for the same individual
http://vivo.mydomain.edu/individual/n4796
http://vivo.mydomain.edu/display/n4796
http://vivo.mydomain.edu/individual?uri=http%3A%2F%2Fvivo.mydomain.edu%2Findividual%2Fn4796

The `IndividualController` is also responsible for handling Linked Open Data requests, and again there are a variety of URLs ways to request them. How will you handle all of these URLs that lead to the same page?

8.6.3.1 The RequestedAction

The `RequestedAction` is how the `Controller` asks the `PolicyStack` whether an action is authorized. Each policy may:

- approve the action (`AUTHORIZED`)
- reject the action (`UNAUTHORIZED`)
- let another policy decide `(INCONCLUSIVE`)

If all policies return `INCONCLUSIVE` , the action is rejected.

Most policies are written to check the class of the `RequestedAction`, and to ignore everything they don't understand, like this:

```
if (!(whatToAuth instanceof DisplayDataPropertyStatement)) {
    return new BasicPolicyDecision(Authorization.INCONCLUSIVE, "Unrecognized action");
}
```

The exception to this is RootUserPolicy, which approves every action if the root user is logged in. So,if you create your own class, its likely that only your policy will approve or reject it.

Something to remember: the `Policy` objects do not have access to the current request. So your `RequestedAction` must carry all of the information that the `Policy` will require to make a decision. In this example, the `Policy` needs to know who is logged in, and which profile page they are requesting.

The {{RequestedAction}} class

```
package edu.cornell.mannlib.vitro.webapp.controller.individual;
import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
import edu.cornell.mannlib.vitro.webapp.beans.Individual;
import edu.cornell.mannlib.vitro.webapp.beans.UserAccount;
/**
 * Ask for authorization to display this individual to this user.
 */
public class DisplayRestrictedIndividualAction extends RequestedAction {
    private final UserAccount user;
    private final Individual individual;
    public DisplayRestrictedIndividualAction(UserAccount user, Individual individual) {
        this.user = user;
        this.individual = individual;
    }
    public UserAccount getUser() {
        return user;
    }
    public Individual getIndividual() {
        return individual;
    }
}
```

8.6.3.2 The Controller

So how does the controller request the action, and what does it do if the action is rejected?

There are a few ways to handle this. If your controller is a sub-class of `FreemarkerHttpServlet`, and if you are willing to accept the default behavior, you can use the `requestedActions()` method. Otherwise, you can use the `FreemarkerHttpServlet.processRequest()` method, or just the `doGet()` method.

Remember, the `IndividualController` needs to deal with several different types of URLs and types of requests. However, it has a method that analyzes the request for you, and creates an `IndividualRequestInfo` object. You can get the information you need from that, as shown in the examples below.

8.6.3.2.1 The `requestedActions()` method

This method is a shortcut for subclasses of `FreemarkerHttpServlet`. Just override this method so it returns an `Actions` object. The framework will check to see if the policies approve this requested action. Here is an example.

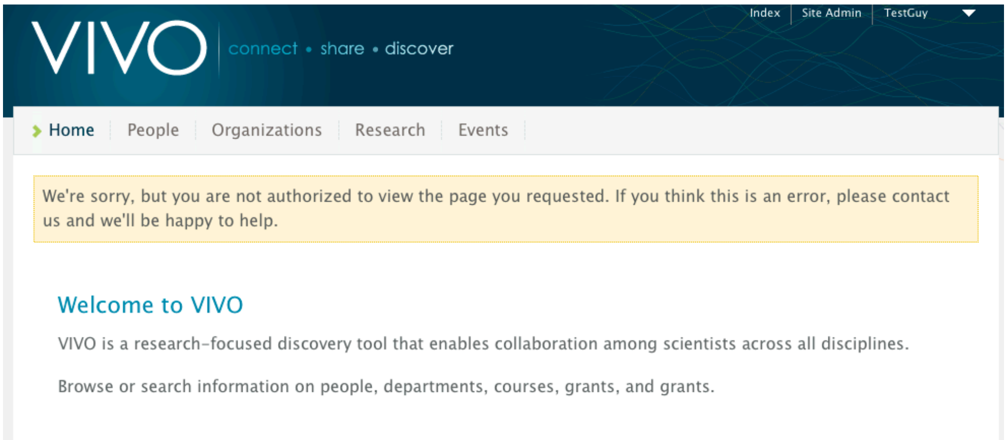
Overriding the {{requestedActions}} method

```
@Override
protected Actions requiredActions(VitroRequest vreq) {
    IndividualRequestInfo requestInfo = analyzeTheRequest(vreq);
    Individual individual = requestInfo.getIndividual();
    UserAccount user = LoginStatusBean.getCurrentUser(vreq);
    return new Actions(new DisplayRestrictedIndividualAction(user, individual));
}
```

8.6.3.2.1.1 What happens if the Policy does not authorize the Action?

If the `PolicyStack` rejects the action, one of two things will happen.

- If the user is not logged in, they will be sent to the login screen. No explanation is offered, but after they log in, the request is repeated.
- If the user is logged in, they will be sent to the home page. A message will appear, like this:



We're sorry, but you are not authorized to view the page you requested. If you think this is an error, please contact us and we'll be happy to help.

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and grants.

8.6.3.2.2 Calling `isAuthorizedToDisplayPage()`

If your controller is not a sub-class of `FreemarkerHttpServlet`, you can accomplish the same result by calling `isAuthorizedToDisplayPage()`. This method takes one or more `RequestedAction` objects, and behaves exactly the same as `requestedActions()` does in a `FreemarkerHttpServlet`.

You must control the code flow yourself, however. If the method returns `false`, your code should immediately return. In that case, the framework has already set the `HttpServletResponse` to redirect as described above.

Simply the code looks like this:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {
    if (!isAuthorizedToDisplayPage(request, response, new MyRequestedAction())) {
        return;
    }
...
```

If you search the VIVO code base, you will find this pattern in several controller classes.

8.6.3.2.2.1 What happens if the Policy does not authorize the Action?

The result is the same as with the `requiredActions()` method.

8.6.3.2.3 For finer control,

In some cases, the default behavior is not wanted. For example, you may want to have your controller display one thing if the action is approved, but display another thing if the action is rejected. In neither case would you want to forward the user to a different page.

In that case, you can call the `isAuthorizedForActions()` method on the `PolicyHelper` class.

```
if (PolicyHelper.isAuthorizedForActions(vreq, new MyRequestedAction())) {
    showAuthorizedResult(request, response);
} else {
    showUnauthorizedResult(request, response);
}
```

8.6.3.2.3.1 What happens if the Policy does not authorize the Action?

That's completely up to you.

8.6.3.3 The Policy

Let's return to the example with the `IndividualController` and the `DisplayRestrictedIndividualAction`. What might the policy look like? Here is a rather silly example. In all likelihood, the actual policy would certainly be more elaborate.

The policy class

```
/* $This file is distributed under the terms of the license in /doc/license.txt$ */

package edu.cornell.mannlib.vitro.webapp.controller.individual;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import edu.cornell.mannlib.vitro.webapp.auth.identifier.IdentifierBundle;
import edu.cornell.mannlib.vitro.webapp.auth.policy.BasicPolicyDecision;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ServletPolicyList;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.Authorization;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyDecision;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyIface;
import
edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
import edu.cornell.mannlib.vitro.webapp.beans.Individual;
import edu.cornell.mannlib.vitro.webapp.beans.UserAccount;

public class PermitProfilesPolicy implements PolicyIface {
    private static final Log log = LogFactory
            .getLog(PermitProfilesPolicy.class);

    @Override
    public PolicyDecision isAuthorized(IdentifierBundle whoToAuth,
            RequestedAction whatToAuth) {
        if (!(whatToAuth instanceof DisplayRestrictedIndividualAction)) {
            return inconclusiveDecision("Only interested in displaying
profiles");
        }
        DisplayRestrictedIndividualAction action =
(DisplayRestrictedIndividualAction) whatToAuth;

        UserAccount user = action.getUser();
        Individual individual = action.getIndividual();
        if (user == null) {
            return inconclusiveDecision("User is not logged in.");
        }
        if (individual == null) {
            return inconclusiveDecision("Not on a profile page.");
        }

        return isAuthorized(user, individual);
    }

    /**
     * This is totally bogus. Presumably you would have more sensible
criteria.
     */
    private PolicyDecision isAuthorized(UserAccount user, Individual
individual) {
        if (individual.getURI().equals(
                "http://vivo.mydomain.edu/individual/n4526")) {
            log.debug("Permit access to " + individual.getLabel());
            return authorizedDecision("I'll let anybody can see this guy.")
;
        } else {
            log.debug("Deny access to " + individual.getLabel());
            return inconclusiveDecision("Some other policy might approve
it, but I won't.");
        }
    }

    /**
     * An AUTHORIZED decision says "Go ahead. Don't need to ask anyone
else".
     */
    private PolicyDecision authorizedDecision(String message) {
        return new BasicPolicyDecision(Authorization.AUTHORIZED,
getClass()
                .getSimpleName() + ": " + message);
    }

    /**
     * An INCONCLUSIVE decision says "Let someone else decide".
     */
    private PolicyDecision inconclusiveDecision(String message) {
        return new BasicPolicyDecision(Authorization.INCONCLUSIVE,
getClass()
                .getSimpleName() + ": " + message);
    }

    //
----------------------------------------------------------------------
    // Setup class
    //
----------------------------------------------------------------------

    public static class Setup implements ServletContextListener {
        @Override
        public void contextInitialized(ServletContextEvent sce) {
            ServletPolicyList.addPolicy(sce.getServletContext(),
                    new PermitProfilesPolicy());
        }

        @Override
        public void contextDestroyed(ServletContextEvent sce) {
            // Nothing to clean up.
        }
    }
}
```

As in the previous example (Creating a VIVO authorization policy - an example (see page 275)), the policy's `Setup` class must be added to `startup_listeners.txt`

8.6.4 The IdentifierBundle - who is requesting authorization?

The policy interface has a single method, and looks like this:

```
public interface PolicyIface  {
    public PolicyDecision isAuthorized(IdentifierBundle whoToAuth, RequestedAction
whatToAuth);
}
```

The nature of `whatToAuth` is covered in Creating a VIVO authorization policy - an example (see page 275) and A more elaborate authorization policy. (see page 283) This page is about `whoToAuth`.

8.6.4.1 The challenge of identity and authorization

A user's level of authorization may depend on a variety of information:

- are they logged in?
- what is their role?
- do they have a profile page?
- what information is in their profile page?

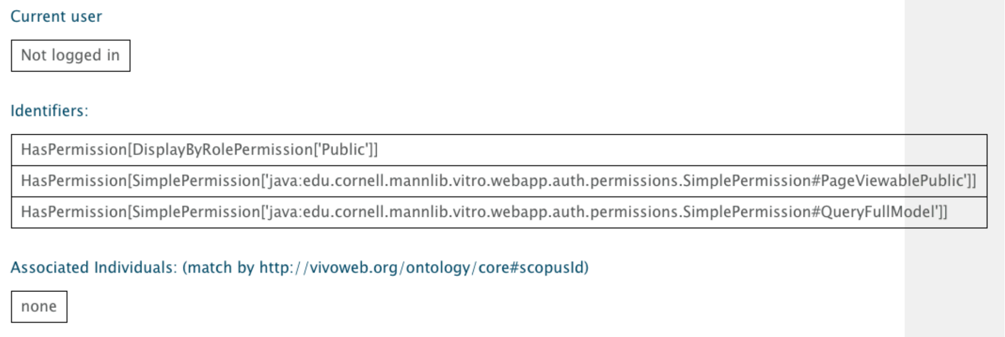
- do they have "proxy authorization" to edit additional pages?

These questions are made more complex because this information is stored in multiple data models. Also, the policy does not have access to the current request or session, so it is not always easy to obtain information.

8.6.4.2 The IdentifierBundle to the rescue

Notice that the `isAuthorized` method receives an argument of the type `IdentifierBundle`. This consists of many `Identifier` objects, and each `Identifier` contains a small piece of information about the current user.

You can see the contents of this bundle (as well as many other things) by directing your browser to `/vivo/admin/showAuth`. This screen shot shows information about an anonymous (not logged in) session:



Current user

Not logged in

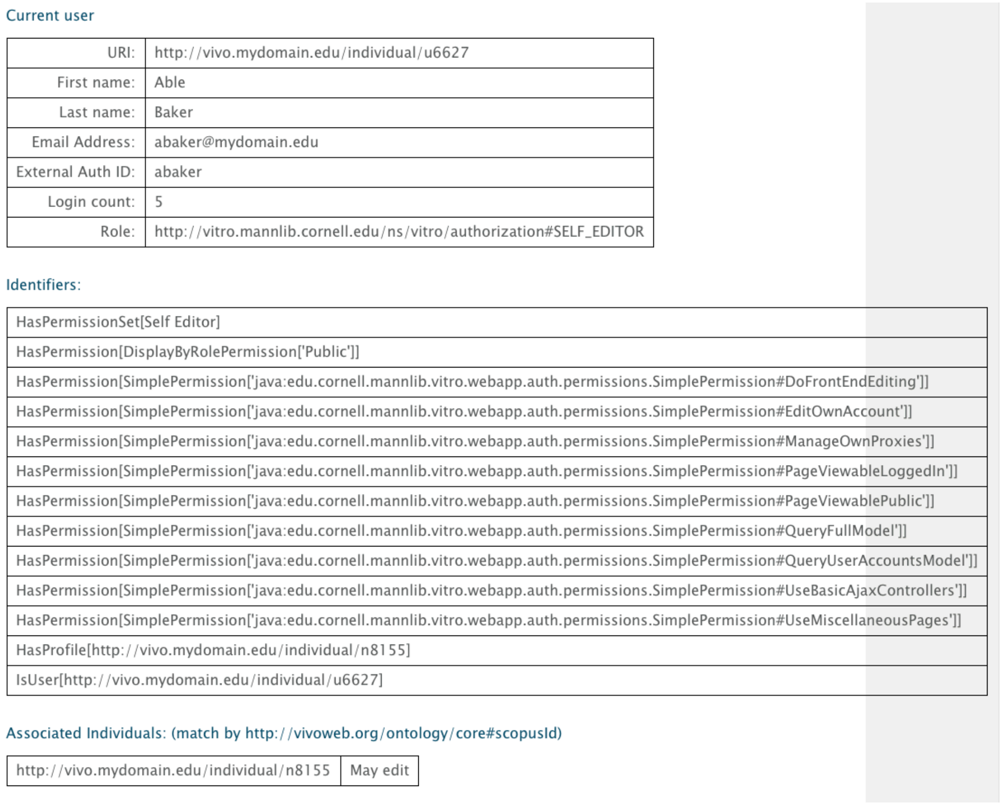
Identifiers:

HasPermission[DisplayByRolePermission['Public']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewablePublic']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryFullModel']]

Associated Individuals: (match by http://vivoweb.org/ontology/core#scopusId)

none

And this one shows information about a user who is logged in as a self-editor.



Current user

URI:	http://vivo.mydomain.edu/individual/u6627
First name:	Able
Last name:	Baker
Email Address:	abaker@mydomain.edu
External Auth ID:	abaker
Login count:	5
Role:	http://vitro.mannlib.cornell.edu/ns/vitro/authorization#SELF_EDITOR

Identifiers:

HasPermissionSet[Self Editor]
HasPermission[DisplayByRolePermission['Public']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#DoFrontEndEditing']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#EditOwnAccount']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#ManageOwnProxies']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewableLoggedIn']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewablePublic']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryFullModel']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryUserAccountsModel']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#UseBasicAjaxControllers']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#UseMiscellaneousPages']]
HasProfile[http://vivo.mydomain.edu/individual/n8155]
IsUser[http://vivo.mydomain.edu/individual/u6627]

Associated Individuals: (match by http://vivoweb.org/ontology/core#scopusId)

http://vivo.mydomain.edu/individual/n8155	May edit

Your policy has access to these `Identifier` objects, and the `Identifier` classes have static methods that make it easier to find the information you want.

For example, in `edu.cornell.mannlib.vitro.webapp.auth.identifier.common.IsUser`

```
String userUri = null;
Collection<String> userUris = IsUser.getUserUris(whoToAuth);
if (!userUris.isEmpty()) {
    userUri = userUris.iterator().next();
}
// null means not logged in.
// Non-null is the URI of the user account.
```

And, in `edu.cornell.mannlib.vitro.webapp.auth.identifier.common.HasProfile`

```
String profileUri = null;
Collection<String> profileUris = HasProfile.getProfileUris(whoToAuth);
if (!profileUris.isEmpty()) {
    profileUri = profileUris.iterator().next();
}
// null means either not logged in, or no profile.
// Non-null is the URI of the profile page.
```

In most cases, the policy is more interested in the URI of the profile page, rather than the URI of the user account. However, either one might come in handy.

It might be worth noting that `HasProfile` and `HasProxyEditingRights` are both subclasses of `HasAssociatedIndividual`. That means that you can easily distinguish between them, or not, according to the needs of your particular policy.

8.7 Adding External Vocabularies

8.7.1 Overview

External vocabulary services are defined in the graph `http://vitro.mannlib.cornell.edu/filegraph/abox/vocabularySource.n3`. You can explore the contents of this graph by navigating to System Admin / Ingest tools / Manage Jena models. Find `vocabularySource.n3` in the list of models. Click Output Model. You will get a file containing the assertions made to define external vocabulary services in your VIVO.

8.8 Search Engine Optimization (SEO)

8.8.1 Overview

VIVO is often used by institutions to highlight and promote the works of their scholars. Promotion works best if the VIVO profile and related pages are easily found by search engines, and considered to be high value by search engines. VIVO provides citation metatags and a site map to help search engines recognize the value of VIVO pages in search results.

VIVO sites can do more to improve boost SEO. See the recommendations from the University of California San Francisco below for additional ideas.

8.8.2 Citation Metatags

Citation meta tags are included on the pages of works. An example from the VIVO sample data is shown below.

Citation metatags regarding a publication

```
<meta tag="citation_author" content="Bogart, Andrew " />
<meta tag="citation_author" content="Roberts, Patricia " />
<meta tag="citation_author" content="Stevens, Emily K" />
<meta tag="citation_date" content="2015" />
```

```
<meta tag="citation_journal_title" content="Journal of Political Rhetoric" />
<meta tag="citation_firstpage" content="1" />
<meta tag="citation_lastpage" content="54" />
<meta tag="citation_volume" content="15" />
<meta tag="citation_issue" content="2" />
```

8.8.3 Sitemap

For better indexing and discoverability of your VIVO installation, a sitemap generator is included - only profile pages are included in the sitemap. To see your sitemap, append `/sitemap.xml` to your VIVO URL.

8.8.4 Additional SEO Considerations

The University of California San Francisco has done considerable work on SEO for research networking systems. They have compared VIVO installations to other systems, and provided guidelines for enhancing SEO. We strongly recommend sites implement as many of their recommendations as possible to boost their SEO for their scholars and their works.

- RNS SEO 2016: How 90 research networking sites perform on Google — and what that tells us https://biomed20.ucsf.edu/2016/08/18/rns-seo-2016/
- RNS SEO: How 52 research networking sites perform on Google, and what that tells us https://biomed20.ucsf.edu/2015/08/14/rns-seo/
- SEO for Research Networking: How to boost Profiles/VIVO traffic by an order of magnitude https://biomed20.ucsf.edu/2014/08/25/seo-for-research-networking/

9 System Administration

- Creating and Managing User Accounts (see page 294)
- Backup and Restore (see page 297)
- Inferences and Indexing (see page 298)
- The Site Administration Page (see page 299)
- The VIVO log file (see page 303)
- Activating the ORCID integration (see page 311)
- Performance Tuning (see page 316)
- Virtual Machine Templates (see page 324)
- Moving your VIVO Instance (see page 325)
- Regaining access to the root account (see page 326)
- Altmetrics Support (see page 327)
- Troubleshooting (see page 328)
- High Availability (see page 332)
- Replicating Ontology Changes Across Instances (see page 333)
- Jena SDB and MySQL setup (see page 334)
- How to mitigate the Log4Shell (CVE-2021-44228, CVSSv3 10.0) vulnerability (see page 336)
- How to mitigate the Spring CVE-2022-22965 vulnerability (see page 336)

9.1 Background

VIVO system administration requires experience in operating systems, Java application administration, Tomcat, MySQL (or triple store or database being used as persistent storage), backup process and Internet security. In addition, familiarity with VIVO data representation (ontology, triples, and RDF formats) is recommended.

9.2 Creating and Managing User Accounts

9.2.1 Overview

In VIVO, the basic functions of browsing and searching are open to anyone. However, if a VIVO user wants to edit items on their profile, view restricted data, or to manage VIVO, they must log in via a User Account.

When a user logs in, they provides their credentials and are associated with a User Account. The credentials are often an Email address and password, but might be different information, depending on how VIVO is configured.

Each User Account has a Role assigned to it. The Role determines how much the user is authorized to do. The lowest Role will permit the user to edit his own profile page. Higher Roles permit editing additional data properties, modifying the ontologies, and administering the VIVO application.

9.2.2 Authentication

9.2.2.1 Internal Authentication

Every VIVO system allows users to log in to an existing User Account by supplying the Email Address and password to the account. Even in an installation that relies on external authentication, there are administrative pages that allow a user to login with Email Address and password.

9.2.2.2 External Authentication

VIVO can be configured to work with an External Authentication system like Shibboleth or CUWebAuth. In that case, the user provides whatever information the External Authentication system requires, and the External Authentication system passes an ID value to VIVO. VIVO recognizes that the user is logged in to the User Account whose "External Authentication ID" field matches that ID.

If a user passes External Authentication, but no User Account matches the ID, VIVO prompts the user to enter his Email Address, First Name, and Last Name, and creates a User Account with that information.

NOTE: To configure VIVO for an External Authentication system, please consult the section entitled 'Enable an External Authentication System (see page 265)'. Note also that the value of the property (the designated External Authentication ID field) must be an **exact match** for the username/email of the user.

9.2.2.3 External-Only Accounts

When creating an account, an administrator may indicate that it is for external authentication only. In that case, no password is assigned to the account, since the External Authentication system manages its own passwords or other credentials.

9.2.3 What is a User Account?

Each User Account is identified by the user's Email address. Each account will have the user's first name and last name, and a role. The account will have additional information, depending on how it is used.

- External Authentication ID – permits logging in by the External Authentication system.
  NOTE: Two User Accounts may not have the same External Authentication ID
- Password – permits logging in by the Internal Authentication system.
- Matching ID – value can be used to associate the User Account with a profile page.

9.2.4 Associating User Accounts with Profile Pages

Each User Account may be associated with an Individual in the VIVO data model. The display page for that Individual is known as the "profile" for that User Account.

A common use of this feature is matching a profile to each member of the campus community. When a user logs in to VIVO, they are directed to their profile page, and are authorized to edit the information on that page.

To associate profiles with user accounts,

1. set selfEditing.idMatchingProperty in runtime.properties to the URI of the datatype property whose values will be used to associate profiles to accounts. A common choice is the university "net id" or other local identifier.
2. For each user that will have a corresponding account, set the value of the idMatchingProperty on the user's profile.
3. Set the value of Matching ID on the user's account to the value on the user's profile.

9.2.5 User Roles

In VIVO there are four user roles that can be assigned: administrator, curator, editor, and self-editor. Future releases will allow VIVO administrators to create additional roles. Permissions provided to roles will determine access options available to user accounts within VIVO. It is important to consider what a new user’s role may be, prior to setting up the new account.

Self-Editor -– The self-editor may create data properties, relationships and entities directly associated to his or her profile.

Editor -– The editor may add, delete and modify entities, object properties and data properties.

Curator -– In addition to performing the tasks of the Editor, the Curator may modify the ontologies, class groups, property groups, and edit site information, including the text displayed on the About page and contact email address.

System Administrator –- In addition to the abilities of the Curator, the Administrator may access the menu management, user accounts, and advanced data tools features. The advanced data tools section include the ingest menu, Add/Remove RDF data, RDF export, SPARQL query, and SPARQL query builder privileges.

9.2.6 The Root User Account

Each VIVO installation has a special User Account, called the root account. The root account has no Role. Nonetheless, the root account is authorized:

- to see all data elements
- to edit all data elements
- to view any page
- to modify the ontologies

Since the root account can do all of these things, it can be particularly useful and particularly dangerous. It can also give you a distorted view of what your VIVO site looks like. Use the root account to create other User Accounts or to access VIVO in emergencies, and use it with deliberation.

The email address for the root account is specified as part of the VIVO installation process.

NOTE: To configure the root account, please consult the Installation Guide, and refer to the section entitled 'Specify Runtime Properties'.

9.2.7 Managing User Accounts

9.2.7.1 Normal workflow

In normal operation, users will receive an Email message when a VIVO account is created for them, when their password is reset by an administrator, or when the Email address on their User Account is changed. One benefit of this is that the administrator does not need to know the user's password, and does not need to tell the user his password.

As noted above, when a new account is created, or when an administrator resets the user's password, the user receives an Email message. The message describes the action that has occurred, and includes a link for the user to click, to set the password on the account.

Note: *User Accounts that are created for External Authentication do not require passwords, so no such link is sent.*

9.2.7.2 Workflow without Email

Email notifications can be disabled by configuring VIVO without a "Reply-To" address. In that case, users are not notified when User Accounts are created or changed.

When creating a new User Account, the administrator must set a password, and must inform the user of the password (unless the account is to be used for External Authentication only). When the user first logs in to the account, he will be prompted to change the password. Resetting the password on an account involves a similar process.

Note: *To disable Email notifications, please consult the Installation Guide, and refer to the section entitled 'Specify Deployment Properties'.*

9.2.7.3 External Authentication

In many VIVO installations, the creation of most User Accounts is simple and routine. A user presents credentials to the External Authentication system, and VIVO creates an account with minimal privilege, prompting the user for name and Email Address. In this case, an administrator may edit such an account to assign a higher Role, if desired.

Alternatively, an administrator may create a User Account, add an External Authentication ID, and assign a high-level Role. When the user log in for the first time, they will already have an account with the desired level of privilege.

9.3 Backup and Restore

There are four components that you will want to backup

1. The VIVO home directory
    a. This holds your site's user accounts and encrypted passwords, along with other authorization and display settings.

   b. Holds the Solr search index. The search index is not vital to a backup, since it can be rebuilt. However, rebuilding the index is time-consuming
   c. Also holds any uploaded image files, and any customized RDF files.
   d. Holds your `runtime.properties` file.

2. The VIVO content store
   a. This holds all of your instance data (people, organizations, etc), as well as any customizations that you entered through the GUI. For sites using SDB, the content store is a relational database and will be backed up using the procedures for the database being used.
3. The VIVO configuration store
   a. In most cases, the VIVO RDF store is held in the VIVO relational database (above), but at some sites it might be in a separate triple-store. The configuration store uses TDB. To back up TDB, shut down Tomcat, and back up the files in `<vivo-home>/tdbModels`
4. The VIVO installation directory
   a. If you have customized the templates or the Java code, you will want to preserve those changes.
   b. At a minimum, this directory contains your `build.properties` file.

9.4 Inferences and Indexing

9.4.1 Recompute Inferences

The inference engine / Reasoner may need to be told to run, and that is achieved by a user with administrative privileges visiting a job specific site.

```
http://vivo.mydomain.edu/RecomputeInferences
```

9.4.2 Re-building the search index

The Solr search may need to have its index re built, and that is achieved by a user with administrative privileges visiting a job specific site.

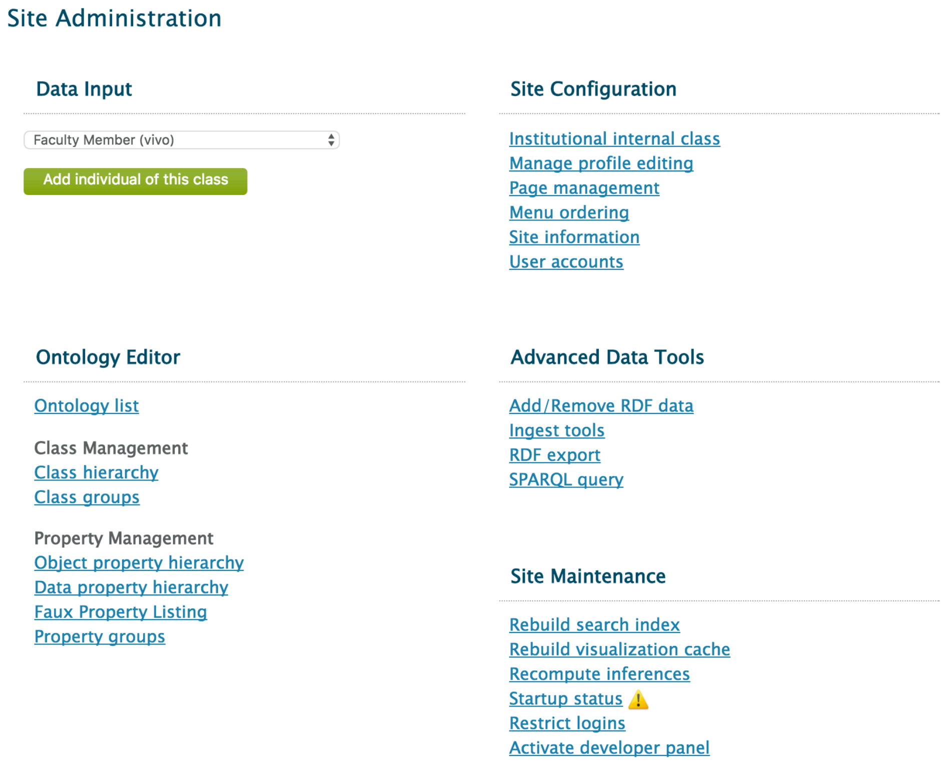
```
http://vivo.mydomain.edu/SearchIndex
```

9.5 The Site Administration Page

9.5.1 Site Administration

Once you are logged into VIVO, you will notice in the upper right hand portion of the page links to "Index" and "Site Admin", alongside a drop-down menu with your name on it, and containing links to "My account" and "Log out".

Once you have logged into VIVO, clicking on the “Site Admin” link takes you to the “Site Administration” page. As an administrator, you will be able to access all five feature and content areas of VIVO: Data Input, Ontology Editor, Site Configuration, Advanced Data Tools, and Site Maintenance. Each is introduced below.



Site Administration

Data Input

Faculty Member (vivo)

Add individual of this class

Site Configuration

Institutional internal class
Manage profile editing
Page management
Menu ordering
Site information
User accounts

Ontology Editor

Ontology list

Class Management
Class hierarchy
Class groups

Property Management
Object property hierarchy
Data property hierarchy
Faux Property Listing
Property groups

Advanced Data Tools

Add/Remove RDF data
Ingest tools
RDF export
SPARQL query

Site Maintenance

Rebuild search index
Rebuild visualization cache
Recompute inferences
Startup status ⚠
Restrict logins
Activate developer panel

9.5.2 Data Input

There are three ways to manually input data into VIVO. 1) on the Site Administration Menu, a new individual of any class may be added directly through the Data Input menu. 2) Selections can be made on many of the pages to add individuals. For example, on user profile pages, users with editing privileges can add, change and remove data. 3) Data can be added as a batch, a collection of RDF triples in a format known to VIVO and expressed in the ontologies known to VIVO.

On the Site Administration page, you can enter a new individual of any type by selecting the type from the drop down menu, and pressing "Add Individual of this class." VIVO will add an individual of the class you have selected, creating a new URI for the individual, and creating an assertion that the individual is a member of the class you have selected. Once an individual has been created, object and data properties may be added for that individual on the page displaying the individual's profile. The object and data properties presented for editing will vary by the type of the individual, in accordance with the ontology;

9.5.3 Ontology Editor

In VIVO, information is identified by references to Unique Resource Identifiers (URIs). URIs can be used by other web pages and applications to locate and retrieve specific chunks of data. The detailed level to which VIVO captures information enables complex relationships among data to be represented.

The VIVO web application is built using RDF "triples" or statements consisting of a subject (known as an individual, item, or entity), a predicate (an object property or a data property) and an object (any individual in VIVO). Subject-predicate-object triples express the relationships among the individuals in VIVO via object properties and support attributes of individuals via data properties.

The first two parts (subject and predicate) of every triple are URIs. An object property triple has the URI of another individual in VIVO its object, while the third element of a data property triple is a data value – typically a text string, number, or date.

Ontology List - VIVO supports keeping an internal list of ontology namespaces and corresponding prefixes to facilitate using external ontologies as well as to help differentiate local ontology additions from VIVO core.

9.5.3.1 Class Management

Individuals in VIVO are typed as members of one or more classes organized and displayed as a hierarchy.

Class hierarchy - The class hierarchy provides a framework to help identify the different types of individuals modeled in a VIVO application. In the Class Hierarchy page, you can edit/add classes, add entities to a class, and add auto links.

Class groups - Class groups are a VIVO-specific extension to support using VIVO as a public website as well as an ontology and content editor. Class groups are a means to organize the classes in VIVO into groups. They represent the facets seen when VIVO is searched (people, activities, events, organizations, etc).

9.5.3.2 Property Management

If classes define what each individual in VIVO is, properties define how that individual relates to other individuals and allow an individual to have attributes of its own. VIVO has two property editors, one for object properties and another for data properties.

Object property hierarchy - Object properties represent the relationship between entities (also known as items or individuals) in VIVO. Object properties can be created and edited from the Object Property Hierarchy.

Data property hierarchy - A data property connects a single subject individual (e.g., a Person or Event) with some form of attribute data. Data properties can be created and edited from the Data property hierarchy link.

Faux property listing - Faux properties are a VIVO-specific extension to allow the same object property to be used in various contexts, with a context specific label and context specific domain and range. A listing of the faux properties in VIVO. You can display the list alphabetically, or organized by base property. The Site Administration page provides a simple view-only listing. See Create and edit faux properties (see page 172) to manage faux properties.

Property groups - Like class groups, property groups are a VIVO-specific extension to support using VIVO as a public website as well as an ontology and content editor.

9.5.4 Site Configuration

This section discusses the site configuration aspects of VIVO. It enables administrators to add or adjust to their institution's site specific details, as well as to manage menus, tabs, and user accounts.

Institutional internal class – set the class that will be used to indicate that individuals are part of your institution. See Create, Assign, and Use an Institutional Internal Class (see page 65)

Manage profile editing – Assign profile editors to individual profiles. Use this feature to allow someone other than the profile owner to edit the owner's profile.

Page management – Create and manage custom pages, as well the presence of pages on menus. See Menu and page management (see page 163)

Menu Ordering – order the menus on the main VIVO navigation banner. See Menu and page management (see page 163)

9.5.4.1 Site Information

The Site information link provides administrators with the capabilities of editing and adding site specific details for that institution's instance of VIVO.

Site Name — Text entered here will be displayed in the browser title bar and bookmark label. It is set to "VIVO" by default.

Contact email address — This field is the email address or listserv that you want the Contact Us form to use. The SMTP host in your configuration file (runtime.properties) must be set for the Contact Us form to work as intended.

Theme — The default theme is "wilma". If you create a new theme (see Creating a custom theme (see page 210)), then it should be available to choose in this drop-down pick list.

Copyright text — Text entered here for a label in the footer for the copyright URL.

Copyright URL — The URL you want the copyright to go to in the footer. It could be your institution's copyright information or the actual institution.

9.5.5 Advanced Tools

The Advanced tools are VIVO's built-in features for data management and export. Please refer to the Advanced Tools section below for detailed instructions.

In addition, many VIVO adopters may require additional information regarding the importing and exporting of RDF data and creating SPARQL queries.

There are several avenues available to acquire guidance with these advanced tools. Information sources such as the VIVO Data Ingest Guide, the W3C's Resource Description Framework model, and the W3C's SPARQL Query Language for RDF, to name a few. Please refer to Appendix A for links and additional resources.

Add/Remove RDF data **–** This tool allows for the manipulation of RDF data in the main model through importing RDF documents for addition or removal.

Ingest tools – A suite of data management tools. See below for a description of each tool.

RDF export - This tool allows for the export of ontology and data in a variety of RDF formats. Options include:

- Export all instance data
- Export a specific ontology such as FOAF, VIVO core, SKOS, etc.
- Export the entire ontology for VIVO

SPARQL query - This tool allows SPARQL select, construct, and describe statements against the main model to be saved in a variety of formats including: CSV, RDF/XML, N3 and more.

9.5.5.1 Ingest tools

Manage Jena Models – This tool allows for the management of the main webapp, as well as separate data models and datasets. The ability to attach separate models to the webapp, load RDF data to a mode, clear statements, and output models as N3 RDF is performed here.

Subtract One Model from Another — This tool allows for the comparison of models for updating information that already exists in VIVO. By subtraction of a current model from a newly constructed model (from the same data source) and vice versa, the additions and subtractions for updating the data are generated.

Convert CSV to RDF — This tool allows for VIVO to read and convert CSV (comma-separated values) and Tab-delimited data into RDF

Convert XML to RDF — This tool allows for VIVO to read and convert well-formed XML into RDF

Execute SPARQL CONSTRUCT — This tool allows for using SPARQL to produce desired RDF from one or multiple source models. This tool is commonly used to map classes and properties to VIVO namespace(s).

Generate Tbox — Tbox statements describe the terms of controlled vocabularies, for example, a set of classes and properties that constitute the ontology. This tool allows for the creation of a Tbox from one or multiple source models.

Name Blank Nodes — This action turns blank nodes, a node in an RDF graph which is not identified by a URI and is not a literal, into nodes with either randomly generated or pattern based URIs.

Smush Resources — This tool allows for using a compression method to distinguish like entities and “Smush” them together based on the specified URI of a property.

Merge Resources — This tool allows two individuals with different URIs to be collapsed into a single URI. Any statements using the “duplicate individual URI” will be rewritten using the “primary individual URI.” If there are multiple statements for a property that can have only a single value, the extra statements will be retracted from the model and offered for download.

Process Property Value Strings — This tool allows for an arbitrary method on a Java class available on the application class path to transform string values of a given property. The method should take a single String as a parameter and return a String.

Change Namespace of Resources — This tool will change all resources in the supplied “old namespace” to be in the “new namespace.” Additionally, the local names will be updated to follow the established “n” + random integer naming convention.

Split Property Value Strings into Multiple Property Values — This tool allows for parsing multiple property values from a single ingested string. This can be used to parse MeSH Terms, controlled vocabulary, and keywords associated with the ingested data.

Execute Workflow — This tool allows for a simple way of scripting actions (specified in RDF) that would otherwise require manual interaction with the ingest tools.

Dump or restore the knowledge base – dump or restore configuration models or content models

9.5.6 Site Maintenance

Rebuild search index – in some situations, you may need to rebuild the SOLR search index. See Inferences and Indexing (see page 298)

Rebuild visualization cache – Large-scale visualizations like the Temporal Graph or the Map of Science involve calculating total counts of publications or of grants for some entity. Since this means checking also through all of its sub-entities, the underlying queries can be both memory-intensive and time-consuming. For a faster user experience, we wish to save the results of these queries for later re-use. To this end we have devised a caching solution which will retain information about the hierarchy of organizations-namely, which publications are attributed to which organizations-by storing the RDF model. We're currently caching these models in memory. The cache is built (only once) on the first user request after a server restart. Because of this, the same model will be served until the next restart. This means that the data in these models may become stale depending upon when it was last created. To avoid restarting the server in order to refresh the cache, administrators can use the Rebuild visualization cache link.

Recompute inferences – in some cases, you may wish to recompute the inferences in VIVO. See Inferences and Indexing (see page 298)

Startup status – shows the messages that were produced during VIVO startup.

Restrict logins – toggles user login. When logins are restricted, only the root user may login

Activate Developer panel – Shows the developer panel from which additional debugging information is available. See Tips for Interface Developers (see page 259)

9.6 The VIVO log file

The VIVO log file contains time-stamped statements intended to help you

- identify the configuration of VIVO,
- monitor the progress of the application, and
- diagnose problems that occur.

The log file is written to the `logs` directory of your Tomcat application. It is usually called `vivo.all.log`, but the name may vary, depending on how your VIVO was installed.

The log file can also be helpful during development and debugging. This is particularly true if the developer takes advantage of the different logging levels.

9.6.1 What does a log message look like?

Here is an example of some code that writes to the log

```
private static final Log log = LogFactory.getLog(WebappDaoSetup.class);
...
log.info(elapsedSeconds + " seconds to set up models and DAO factories");
```

and here is the resulting line in the log:

```
2012-11-15 12:20:37,406 INFO [<span class="confluence-link">WebappDaoSetup</span>] 3
 seconds to set up models and DAO factories
```

The log holds the time that the statement was written, the severity level of the message,
the name of the Java class that wrote the statement, and the contents of the statement itself.

Writing exceptions to the log can be tricky: check out this page on Writing Exceptions to the Log (see page 308)

9.6.2 What is the right level for a log message?

Each log message has an output level (sometimes known as a severity level).
The most common levels are DEBUG, INFO, WARN, ERROR.
Each level conveys a sense of how important the message is.

ERROR	Serious errors which need to be addressed and may result in unstable state.
WARN	Runtime situations that are undesirable or unexpected, but not necessarily "wrong", especially if the system can compensate; "almost" errors.
INFO	Interesting runtime events; routine monitoring information. Commonly used to describe how the system starts up, or changes that are worth noting as the system runs.
DEBUG	Used by developers when debugging their code. These messages will not appear in the log unless specifically enabled (see below)

The logging framework also supports the levels of `FATAL` for very serious errors,
and `TRACE` for verbose debugging messages, but these are much less commonly used.

9.6.3 Setting the output levels

9.6.3.1 Production settings

The output levels for VIVO are determined by a file called `[vitro-core]/webapp/config/log4j.properties`

This file sets the general output level to `INFO`, which means that messages at the `INFO` level or higher will be written to the log. Messages at `DEBUG` or lower will not be written to the log.

The file also sets higher output levels for some classes that are otherwise too chatty with their log messages. So for example, the `StartupStatus` class is assigned an output level of `WARN`. This means that messages at the `WARN` level or higher will be written to the log, and messages at `INFO` or lower will not.

9.6.3.2 Developer settings

Developers can make temporary changes to these settings by creating a file called `[vitro-core]/webapp/config/debug.log4j.properties`

When VIVO is rebuilt, the settings in this file will be used instead of the settings in the default file. A developer will commonly change the output level of the classes or packages he is currently working on, using this file.

The debug settings file is ignored by Git. As a result it remains unique to the individual developer, and can be changed without concern.

The debug settings file should not be present in a VIVO that is being built for production use.

9.6.3.3 Changing levels while VIVO is running

You can change the log levels for individual Java classes while VIVO is running.

Direct your browser to `[vivo]/admin/log4j.jsp` This page requires that you log in to VIVO as an administrator.

This page shows a list of all Java classes with active Logger components. Each class has a drop-down list that allows you to set the log output level for that class. Select the level(s) you want, and scroll to the bottom of the page to click the button labeled `Submit changes to logging levels`. The change is effective immediately.

This feature should be used with care. A log level of `DEBUG` can significantly slow down some Java classes, and can result in very large amounts of output to the log of a busy system.

Note: The `log4j.jsp` *page shows only the classes with **active** Loggers. This means that you can't set use this page to set the output level of a class prior to the first time it is used. Java loads classes dynamically, and until the class is loaded, it does not have an active Logger.*

9.6.4 Customizing the logging configuration

9.6.4.1 Overview

VIVO uses the Log4J package for logging status messages. VIVO is shipped with a configuration file that sets up the logging properties, so the VIVO log is written to `vivo.all.log` in the `[tomcat]/logs` directory. Most sites find this default configuration suitable when they start out, but often as people become more experienced with VIVO, they prefer to change the logging options.

9.6.4.2 The default configuration

The configuration file is found at `[vitro]/webapp/config/log4j.properties` . The file looks something like this:

```
 1  log4j.appender.AllAppender=org.apache.log4j.RollingFileAppender
 2  log4j.appender.AllAppender.File=$${catalina.home}/logs/${webapp.name}.all.log
 3  log4j.appender.AllAppender.MaxFileSize=10MB
 4  log4j.appender.AllAppender.MaxBackupIndex=10
 5  log4j.appender.AllAppender.layout=org.apache.log4j.PatternLayout
 6  log4j.appender.AllAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{1}] %m%n
 7  
 8  log4j.rootLogger=INFO, AllAppender
 9  
10  log4j.logger.edu.cornell.mannlib.vitro.webapp.startup.StartupStatus=WARN
11  log4j.logger.edu.cornell.mannlib.vitro.webapp.dao.jena.pellet.PelletListener=WARN
12  log4j.logger.org.springframework=WARN
13  log4j.logger.com.hp.hpl.jena.sdb.sql.SDBConnection=ERROR
```

(The listing above has been abridged for clarity. Comments have been removed, as have some repetitious lines.)

The file creates an "appender", which tells Log4J where to write the log messages, and how to manage them. It creates a "root logger" which will set the default properties for all logging: using the named appender and omitting any messages that are lower than INFO level. Finally, it overrides the logging threshold level for some special classes and packages.

In more detail (by line numbers):

> (1) Use a RollingFileAppender. This will write messages to the named file, until the file becomes too large. Then the accumulated messages are "rolled over" to a backup file, and logging continues.
>
> (2) Specify the name and location of the log file. During the build process, `${webapp.name}` will be replaced by `vivo`, or whatever you have chosen as the name of your webapp. When VIVO starts, Log4J will replace `$${catalina.home}` with the value of the system property named `catalina.home`. This is the Tomcat home directory.
>
> (3) Files will roll over when they reach 10 MegaBytes of content.
>
> (4) No more than 10 files will be kept
>
> (5, 6) The message layout is determined by this pattern. It consists of the date and time, the severity of the message, the name of the class writing the message, and the message itself (followed by a linefeed).
>
> (8) The root logger, and by default all loggers, will write to this appender. Only messages with a level of INFO or higher will be written to the log. That is, messages with levels of DEBUG or TRACE will not be written.

(10, 11) Override the defaults for these classes. The write too many INFO messages, so we restrict them to WARN or higher.

(12) Override the default for the entire package of `org.springframework`

(13) Don't show messages from `com.hp.hpl.jena.sdb.sql.SDBConnection` unless they are ERROR or FATAL.

9.6.4.3 Writing some messages to a special log

Here is an example of how to override the defaults for particular classes in VIVO. In this example, the messages associated with rebuilding the search index are to be written to a special log file. The messages about re-inferencing are also to be written to that file.

The lines below can be added to the end of the default configuration:

```
log4j.appender.SpecialAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.SpecialAppender.DatePattern='.'yyyy-MM-dd
log4j.appender.SpecialAppender.File=/usr/local/vivo/logs/inference_and_indexing.log
log4j.appender.SpecialAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.SpecialAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{1}] %m%n
log4j.logger.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexBuilder=SpecialAppender
log4j.logger.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexWorkerThread=SpecialAppender
log4j.logger.edu.cornell.mannlib.vitro.webapp.reasoner.ABoxRecomputer=SpecialAppender
```

Here we define a second appender, and tell three particular Java classes to use that appender.

Again, by line numbers:

(15, 16) Use a DailyRollingFileAppender. Unlike the RollingFileAppender, this log file will roll over at midnight every day. There is no maximum number of files.

(17) The log file will be `/usr/local/vivo/logs/inference_and_indexing.log`. At midnight, the file will be renamed to `inference_and_indexing_log.2013-06-21` (for example).

(18, 19) The layout of the message is the same as for the main log file

(20, 21, 22) These three classes will write to the new appender.

Notice that the log messages for these classes will now be written both to the main log file and to this special file. By default, the appenders are "added" to the classes where they are specified. If you want these classes to only write to the special file, you must turn off the "additivity" property of those classes, as shown below:

```
log4j.additivity.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexBuilder=false
```

```
log4j.additivity.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexWorkerThread=false
log4j.additivity.edu.cornell.mannlib.vitro.webapp.reasoner.ABoxRecomputer=false
```

9.6.4.4 More information

Log4J is a very powerful and flexible framework. Many different options are available through the use of appenders, layouts, and filters. For more information, you may want to consult

- The Log4J manual[126] – a compact discussion of the many aspects of Log4J.
- The Log4J API documentation[127]
- The documentation of the Log4j properties file[128]

9.6.5 Writing Exceptions to the Log

9.6.5.1 Not the Right Way

This is not a good way to handle an exception:

```
} catch(Exception e) {
}
```

An exception occurred, but we ignored it. Don’t do this. Please.

This isn't very good either (although, to be fair, it is better than a kick in the head):

```
} catch(Exception e) {
  e.printStackTrace();
}
```

In Vivo/Vitro the stack trace is printed to catalina.out instead of vivo.all.log. In the Vivo Harvester it is printed to standard out (System.out). It has no timestamp and no source information, so we can’t correlate it with other messages in the log. Were any other messages produced by the same request? We’ll never know.

9.6.5.2 Declaring a Logger

In Vivo and Vitro, we use Apache Commons Logging. Create a logger in your Java code with a couple of imports and a static variable:

---

[126] http://logging.apache.org/log4j/1.2/manual.html

[127] http://logging.apache.org/log4j/1.2/apidocs/

[128] http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PropertyConfigurator.html

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyClass {
    private static final Log log = LogFactory.getLog(MyClass.class);
    ...
```

In the Vivo Harvester, we use Simple Logging Facade 4 Java. Create a logger in your Java code much like ACL:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyClass {
    private static Logger log = LoggerFactory.getLogger(MyClass.class);
    ...
```

9.6.5.3 Bad, Better, Good

So, if this isn't good, how can we improve on it?

```
} catch(Exception e) {
}
```

This is better. We’re still ignoring it, but we could stop ignoring it just by raising the logging level:

```
} catch(Exception e) {
  log.debug(e, e);
}
```

This is better still. Here is a clue as to why we’re ignoring the exception.

```
} catch(Exception e) {
  // This happens if the model data is bad - it's not important
  log.debug(e, e);
}
```

What if we do want to write the exception to the log? What's the right way to do it?

Not like this, for reasons mentioned earlier:

```
} catch(Exception e) {
  e.printStackTrace();
}
```

This is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

If you have an idea of why a certain exception might be occurring, this would be the best:

```
} catch(IllegalStateException e) {
  log.error("One of the flay-rods has gone out of skew.", e);
} catch(Exception e) {
  log.error(e, e);
}
```

But alas, sometimes no useful message occurs to us.

9.6.5.4 Whoops

Unlike some other logging frameworks (Log4J, for example) Apache Commons Logging won't check to see whether your first argument is an exception. Instead, it just converts it to a String and prints it to the log.

So, this probably doesn't do what you wanted:

```
} catch(Exception e) {
  log.error(e);
}
```

It logs the class of the exception, and the message in the exception, but it doesn't write the stack trace. That's why this is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

This way, the Exception class and it's message are written to the log twice, but that's a small price to pay – at least you get the stack trace in the log as well.

And this is best:

```
  } catch(ExpectedTypeAException e) {
    log.error("Some informative message explaining why TypeA might occur", e);
  } catch(ExpectedTypeBException e) {
    log.error("Some informative message explaining why TypeB might occur", e);
  } catch(Exception e) {
    log.error("Some informative message explaining that an unexpected error occurred",
e);
  }
```

Because you get to provide more information, you don't write anything twice, and you do get the stack trace.

9.7 Activating the ORCID integration

9.7.1 Overview

VIVO contains code that will converse with the ORCID registry through its API. When this conversation is enabled, a VIVO user can authoritatively confirm his ORCID iD in VIVO, and cite his VIVO page in his ORCID record as an external identifier.

In order to activate the VIVO-ORCID integration, your organization must have a membership in ORCID. You may then register your VIVO installation as a client application, and obtain the credentials needed for that connection.

Once you have the credentials, you can enter them in the runtime.properties file and restart VIVO.

You may want to start by obtaining credentials for ORCID's sandbox API. This will let you see how the integration appears. If you have made local modifications to VIVO, you will want to ensure that they do not interfere with the integration before going into production.

Once you are satisfied that the integration is working as expected, you can apply for credentials on ORCID's production registry.

9.7.2 Video tutorial

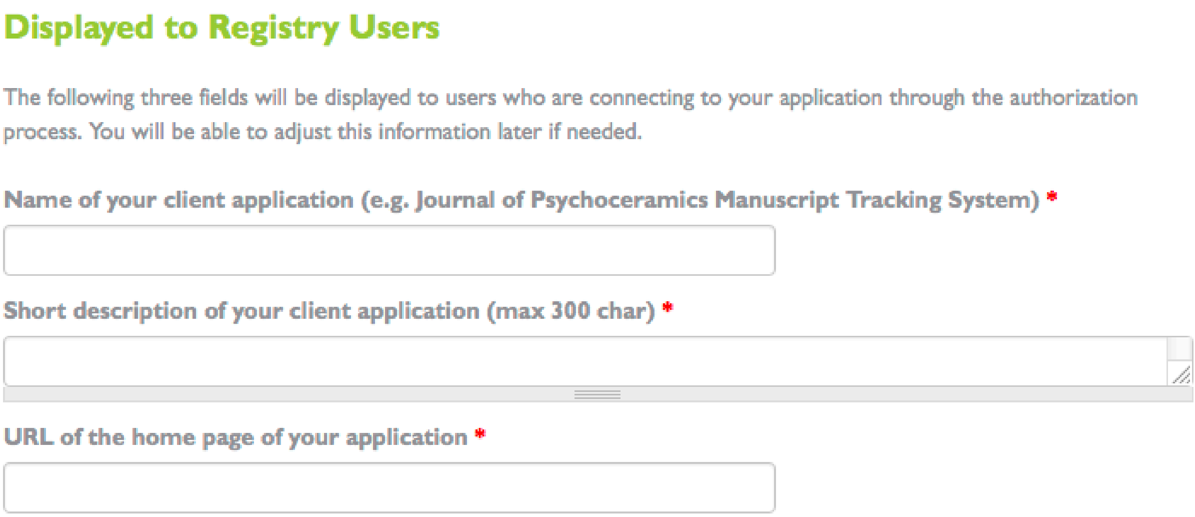
Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

http://youtube.com/watch?v=iCLL1Hm--sw

9.7.3 When applying for credentials

9.7.3.1 Informing the users

The user must grant authorization before VIVO can read or write to their ORCID record. Some of the text they see will come from your credentials. Notice this section of the application:



Displayed to Registry Users

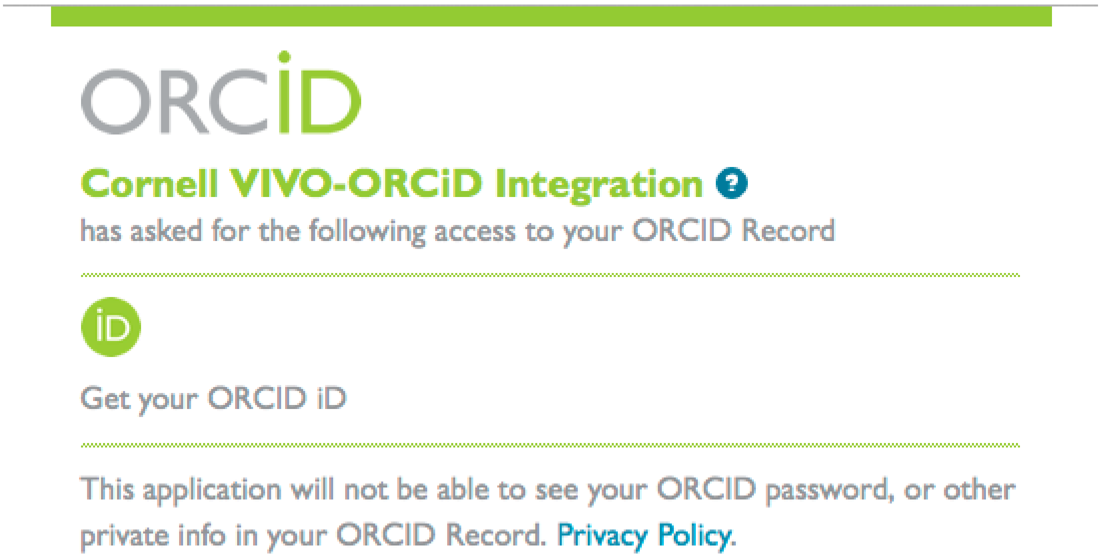
The following three fields will be displayed to users who are connecting to your application through the authorization process. You will be able to adjust this information later if needed.

Name of your client application (e.g. Journal of Psychoceramics Manuscript Tracking System) *

Short description of your client application (max 300 char) *

URL of the home page of your application *

The name of your client application will be displayed to the user as they use the integration screens. Here is an example, where the name of the client application is "Cornell VIVO-ORCID Integration".



ORCID

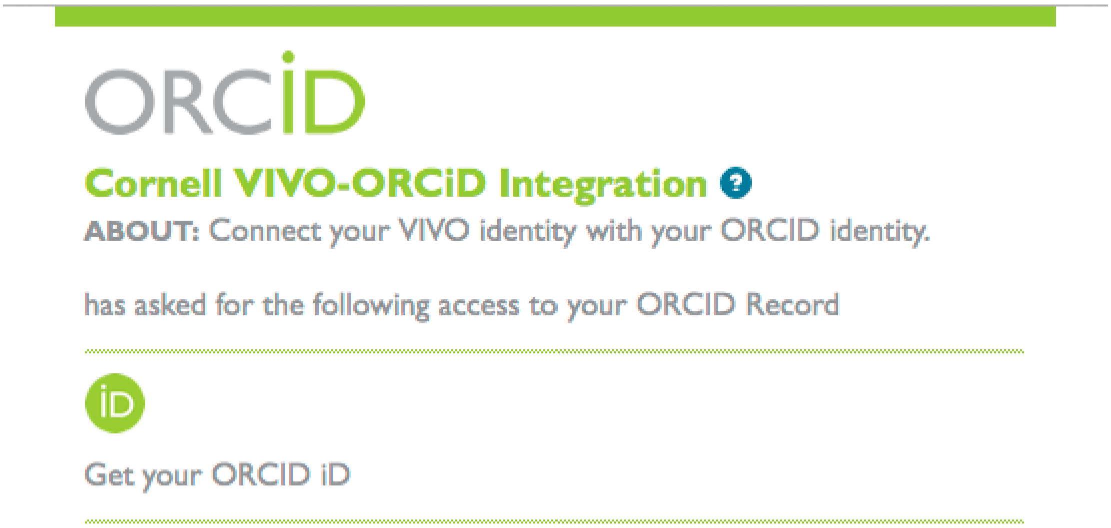
Cornell VIVO-ORCiD Integration

has asked for the following access to your ORCID Record

Get your ORCID iD

This application will not be able to see your ORCID password, or other private info in your ORCID Record. Privacy Policy.

If the user clicks on the question mark, they will see the short description of your client application. In this example, the short description is "Connect your VIVO identity with your ORCID identity."



ORCID

Cornell VIVO-ORCiD Integration

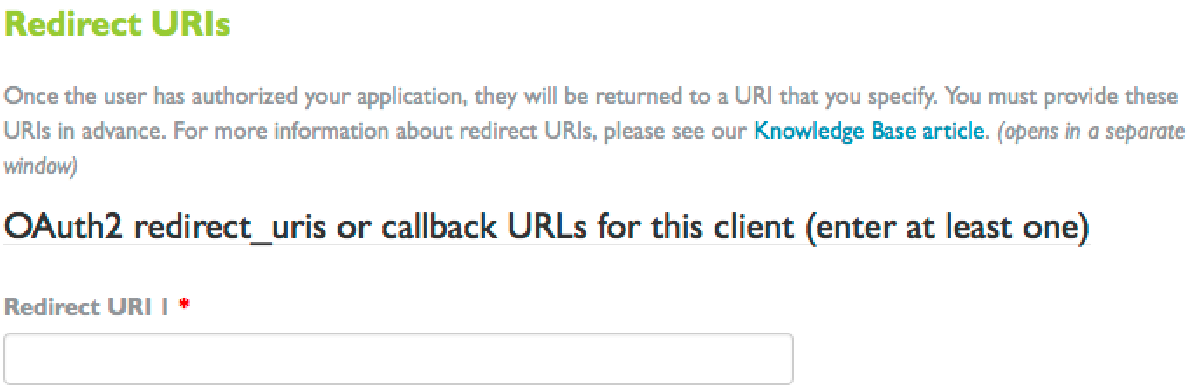
ABOUT: Connect your VIVO identity with your ORCID identity.

has asked for the following access to your ORCID Record

Get your ORCID iD

9.7.3.2 Connecting to your application

Once the user logs in to their ORCID account, and grants authorization to your application, the ORCID pages will transfer control of the session back to VIVO. In order to do that, it needs to know where your application is located. Notice this section of the application:



Redirect URIs

Once the user has authorized your application, they will be returned to a URI that you specify. You must provide these URIs in advance. For more information about redirect URIs, please see our Knowledge Base article. *(opens in a separate window)*

OAuth2 redirect_uris or callback URLs for this client (enter at least one)

Redirect URI 1 *

You may provide just the domain of your application, such as `http://vivo.mydomain.edu`.

9.7.4 Configuring VIVO

To converse with ORCID, VIVO requires these values in the `runtime.properties` file.

Property name	`orcid.clientId`

Description	The Client ID from your ORCID credentials When your application for credentials is accepted, you will receive a Client ID to be used in communications with the API. If you apply for sandbox credentials first, and then production credentials, you will likely receive two different Client IDs.
Default value	NONE
Example value	`0000-0012-0661-9330`

Property name	`orcid.clientPassword`
Description	The Client Secret from your ORCID credentials When your application for credentials is accepted, you will receive a Client Secret to be used in communications with the API. If you apply for sandbox credentials first, and then production credentials, you will likely receive two different Client Secrets.
Default value	NONE
Example value	`103de999-1a37-400c-309f-2094ba72c988`

Property name	`orcid.webappBaseUrl`
Description	The base URL for your VIVO application, as seen from outside. VIVO will use this to construct a callback URL that the ORCID API can use to return control to VIVO. The actual callback URL will be the string you provide here with the suffix of `/orcid/callback` added at the end.
Default value	NONE
Example value	`http://vivo.mydomain.edu` `http://some.domain.edu/vivo/`

Property name	`orcid.apiVersion`

Description	The version of ORCIDs API protocol that VIVO will expect. Versions `1.0.23, 1.2, or 2.0`
Default value	NONE
Example value	`2.0`

Property name	`orcid.externalIdCommonName`
Description	The label used to describe a VIVO profile page If the user authorizes the addition of their VIVO profile page to their ORCID record, it will appear as an "external ID", with this label
Default value	NONE
Example value	`VIVO profile page at Great Western University`

Property name	`orcid.api`
Description	The entry point for ORCID's public API. This changes, depending on whether you are using the sandbox API or the production API.
Default value	NONE
Example value	`sandbox` `release`

9.8 Performance Tuning

9.8.1 SDB - MySQL Tuning

By default, MySQL has reasonable defaults for a regular RDBMS application. However, SDB has a slightly unusual database layout - it has very few tables, some of which grow quite large, very quickly. Whilst the SDB code is well optimised for the majority of cases, to get the best performance, you should tune MySQL to take into account the table, index and join sizes.

9.8.1.1 Version Recommendation

It is recommended that you use 5.5 or later of MySQL (or the MariaDB equivalent).

9.8.1.2 MySQL DB Engine

It is recommended that you use innodb with the barracuda file format. You should also configure MySQL to use a file for each table.

```
innodb_file_per_table = 1
innodb_file_format = barracuda
```

9.8.1.3 MySQL Buffers

Although this won't affect an initial query, having large buffers for the indexes will help query performance once they have been warmed.

```
join_buffer_size = 32M
read_rd_buffer_size = 32M
innodb_buffer_pool_size = 1536M
```

9.8.1.4 Temporary Tables

SDB can generate some large joins, and by default anything over 16MB will be spooled to disk. This can slow large queries down dramatically. To avoid this, increase the temporary table sizes.

```
max_heap_table_size=256M
tmp_table_size=256M
```

9.8.2 Additional Performance Tips

9.8.2.1 What is performance?

Performance can mean different things to different sites including the length of time it takes to render a large page (e.g., a person with 800 - 1500 publications), to display a visualization, to load new data, to regenerate the search index or recompute inferences, or to generate an export of RDF data.

9.8.2.2 What kind of performance is normal? How do I know if I have a problem?

This section gives some very rough guidelines for determining whether your VIVO is performing similarly to established production installations on typical modern server hardware or virtual machines. The numbers below assume that VIVO is otherwise idle; that is, not loaded with concurrent public page requests or performing other background operations.

9.8.2.2.1 Individual page display

The time it takes to render an individual page can vary significantly depending on the types of data involved. The page for a person with many publication citations will take longer to render than one with simple links to other individuals. As a very general rule, your VIVO should be able to handle around 100 data items (properties) per second when displaying an individual page. Thus, if the page for a person with 500 publication links displays in five seconds, there may be relatively little room for performance tweaking short of caching the entire page. If the page takes 50 seconds to appear, there is very likely a serious performance bottleneck somewhere in the installation or a hardware deficiency that needs to be addressed.

9.8.2.2.2 RDF loading

Loading RDF through VIVO is slower than inserting it directly into the triple store because VIVO performs additional operations such as inference and search index maintenance as the data are changed. You should still expect to see at least several hundred triple insertions per second.

9.8.2.2.3 Inference recomputation and search index rebuilding

These operations are important for VIVO installations that modify data directly in the triple store instead of adding or removing RDF through VIVO. You should expect inference recomputation to average about 20-25 milliseconds per individual. (You can find your values in `vivo.all.log`.) Search index rebuilding is typically faster, on the order of 10 ms per individual.

9.8.2.3 Tools for measuring performance

Members of the VIVO community have found the following tools helpful in testing and measuring a site's performance:

- Google Analytics. Records some basic performance metrics in the Behavior > Site Speed section, such as average page load time.

- JMeter. Generates simultaneous connections for testing of performance under real-world production loads.
- New Relic. Software analytics suite including JVM and MySQL monitoring.

9.8.2.3.1 Testing without local modifications

Local code modifications – especially custom list views and filter policies – can introduce inefficiencies that lead to poor performance. Similarly, code under development may contain performance regressions or new features that have not yet been optimized. If you have made any such modifications or are using pre-release code, it is important to test performance when your VIVO database is used with an official VIVO release. If the observed performance differs significantly from that exhibited by a modified version, the modifications are suspect.

9.8.2.4 Tuning for improved performance

9.8.2.4.1 Memory

Ensure that that Java JVM for your VIVO has been allocated sufficient memory (heap space). This is a critical element of the installation process, as the default Java heap setting will cause VIVO to run extremely slowly. A production VIVO installation should typically be allocated several gigabytes of heap space.

Additionally, ensure that your server has enough memory to support the heap space you have allocated. Otherwise, data may be swapped to disk, which can seriously degrade performance. On a server that runs only VIVO, the available memory should be about double the Java heap space.

9.8.2.4.2 Server connections

A production VIVO installation often involves an Apache web server, the Tomcat servlet container, and a MySQL database server. The numbers of available connections between each of these servers should be set to prevent unnecessary bottlenecks. Thus, the maximum number of database connections should slightly exceed the number of possible concurrent Tomcat threads, which should in turn exceed the number of simultaneous Apache worker threads or child processes.

9.8.2.4.3 MySQL configuration

Data display in VIVO often depends on complex SPARQL queries that, when using the default SDB triple store, are translated into similarly complex SQL queries. Tuning the MySQL database server can significantly increase performance. There are a number of tools available for assisting with this process, such as mysqltuner.pl[129] (https://github.com/rackerhacker/MySQLTuner-perl). There are also a few typical parameters that often require adjustment.

---

[129] http://mysqltuner.pl

9.8.2.4.4 In-memory temporary tables

The nature of the SQL queries generated by the triple store often requires the generation of temporary tables. Ideally these temporary tables will remain in memory; if they exceed the threshold where MySQL writes them to disk, this can result in serious slowdowns. Depending on the amount of data in your VIVO and your server's available memory, you may need to increase the size limit for in-memory temporary tables.

Consult the MySQL documentation for the parameters

- tmp_table_size
- max_heap_table_size

9.8.2.4.5 Key buffer size

If your VIVO database uses MySQL's traditional MyISAM storage engine, consult the documentation for the key_buffer_size parameter. Increasing this value can yield significant performance benefit.

9.8.2.4.6 InnoDB buffer pool size

If your VIVO database uses MySQL's newer InnoDB storage engine, consult the documentation for the innodb_buffer_pool_size parameter. Setting this value as large as possible given available memory will improve performance.

9.8.2.4.7 Transaction logging

Changing MySQL's transaction logging settings can lead to dramatic improvements to the speed at which triples are added to or removed from the database. For more details, see „Writing the MySQL transaction log" here: MySQL tuning, and troubleshooting (see page 320)

9.8.2.4.8 HTTP caching

If VIVO's dynamically-generated pages do not exhibit acceptable load times, you may wish to enable HTTP caching. See Use HTTP caching to improve performance (see page 322). With this configuration, subsequent requests for pages whose contents have not changed will result in those pages being served directly from a cache instead of being regenerated from data in the triple store.

9.8.2.4.9 Alternative triple stores

While VIVO is tested with and configured by default to use Jena's SDB triple store with the MySQL database server, VIVO also includes support for TDB and Virtuoso as well as the ability to connect via HTTP to a SPARQL 1.1-complaint endpoint. Use of a different store may yield performance improvements, offer additional possibilities for performance tuning, or enable features such as clustering and load balancing. In addition, configuring SDB to use a database server other than MySQL may offer advantages for your installation. Note that some of the SPARQL queries in the list views (see page 188) employed by VIVO in page rendering have been optimized for SDB/MySQL with substitution of UNION for OPTIONAL. These queries should be modified for optimum performance with other stores that do not exhibit the same quirks.

9.8.2.4.10 Misbehaving robots

In some cases, poor VIVO performance has been traced to search engine robots that either ignore or misread directives in VIVO's robots.txt file, or which issue requests for large pages at a rate that greatly exceeds the demand otherwise encountered in typical production use. If the search engine in question is not critical to VIVO's visibility, it may be advisable to restrict access to the associated robots. In some situations, institutional search appliances are responsible for the excessive server load. Here, discussions with local IT staff may be warranted.

9.8.3 MySQL tuning, and troubleshooting

9.8.3.1 Tuning MySQL

From Stony Brook –

By popular request, I've been asked to re-send information about the MySQLTuner tool. It helped give us feedback on several key mysql tuning parameters. And it gives suggestions on settings that may help your system run more efficiently, and thus your VIVO run a little bit faster.
The mysqltuner.pl[130] script can be found at:
https://github.com/rackerhacker/MySQLTuner-perl

From Mark at Griffith Uni -
We use an enterprise hosted MySQL ie. remote to our vivo server via gigabit ethernet. In this configuration we have found MySQL to be a real performance bottleneck. Here are some parameters that we have found it worthwhile experimenting with:
innodb_flush_log_at_trx_commit=2

- this resulted in about a 3x speedup (especially for big ingests)

tmp_table_size

max_heap_table_size

key_buffer_size (needed because many of our queries include a group or sort)

9.8.3.1.1 Writing the MySQL transaction log

MySQL allows you to control its logging behavior, using the the `innodb_flush_log_at_trx_commit` parameter. On some systems, changing the value of this parameter can dramatically improve performance.

Using the default setting, the log is written to the file buffer and the buffer is flushed to disk at the end of each transaction. This is necessary to insure full ACID compliance, but the overhead is substantial. Most of VIVO is not transaction-oriented: each statement is added or deleted in its own transaction. So the default setting means that a physical write to disk is required for each new RDF statement.

---

[130] http://mysqltuner.pl/

Setting `innodb_flush_log_at_trx_commit` to 0 or 2 will greatly improve throughput, while adding a minimal level of risk to the data. Under some circumstances, with some settings, up to one second of transactions can be lost. Most VIVO installations will find this to be an acceptable level of risk.

setting	meaning	worst case risk
1 (default)	Write the log after each transaction. Flush to disk after each transaction.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose transactions in progress.
2	Write the log after each transaction. Flush to disk once per second.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose one second of transactions.
0	Write the log once per second. Flush to disk once per second.	If MySQL crashes, lose one second of transactions. On power failure or system crash, lose one second of transactions.

This page provides full details regarding `innodb_flush_log_at_trx_commit`: http://dev.mysql.com/doc/refman/5.1/en/innodb-parameters.html#sysvar_innodb_flush_log_at_trx_commit

9.8.3.1.2 Setting the MySQL query cache size

Increasing the MySQL query cache size will likely translate into improved VIVO performance in that once large pages have been fetched once, they're typically quite a bit faster to load on later fetches.

9.8.3.1.3 Tracing back from SQL to SPARQL

If we identify particularly slow SQL queries, we can try to trace them back to SPARQL queries in the code and look for optimizations to those queries or attempt to solve the problem in a different way.

One approach is to watch the status of the MySQL query process during slow queries or page rendering to see what it's doing and/or do an EXPLAIN SELECT on the generated SQL.

9.8.3.1.4 Regenerating MySQL indexes

If performance is abysmal on a simple query, check for missing or corrupted MySQL indexes that may cause the query engine to do full table scans.

9.8.3.1.5 TCMalloc and MySQL

Interesting GitHub blog post (https://github.com/blog/1422-tcmalloc-and-mysql) describing debugging MySQL performance issues, and using tools like the open source Percona Toolkit[131] and the Google-contributed TCMalloc from gperftools[132].

9.8.4 Use HTTP caching to improve performance

As a VIVO implementation grows in size and tracks more and more scholarly activity, profile pages can be pulling in hundreds of relationships to render the page, which results in more data being retrieved from the underlying triple store and longer page load times. For example, a profile page for a faculty member with hundreds of publications, which isn't uncommon, can lead to multiple second page loads.

Instead of querying the database each time a page is loaded, a cached version of the page can be served, provided the user is not logged in. VIVO supports HTTP caching directly. To enable, uncomment the "http.createCacheHeaders = true" line in runtime.properties:

runtime.properties

```
# Tell VIVO to generate HTTP headers on its responses to facilitate caching the
# profile pages that it creates.
#
# For more information, see this wiki page:
# https://wiki.duraspace.org/display/VIVO/Use+HTTP+caching+to+improve+performance
#
# Developers will likely want to leave caching disabled, since a change to a
# Freemarker template or to a Java class would not cause the page to be
# considered stale.
#
  http.createCacheHeaders = true
```

VIVO will now generate eTags for caching, which are stored in VIVO's Solr index. More information is available from Ted Lawless, who originally demonstrated the eTag method, here[133].

Next, enable mod_cache in Apache by uncommenting LoadModule lines in httpd.conf:

httpd.conf

```
LoadModule cache_module modules/mod_cache.so
LoadModule cache_disk_module modules/mod_cache_disk.so
```

---

[131] http://www.percona.com/doc/percona-toolkit/2.1/
[132] http://code.google.com/p/gperftools/
[133] https://lawlesst.github.io/notebook/vivo-caching.html

and adding the following configuration lines to httpd.conf or in its own .conf file within Apache's conf.d directory:

mod_cache.conf

```
#The default expire needs to be 0 in a self-editing environment so that E-Tags can be
reverified.
#Requests to cached URLs that haven't expired will never reach the VIVO web
application.
#

<IfModule mod_cache.c>
     CacheRoot /var/cache/apache2
     CacheEnable disk /display
     CacheEnable disk /individual
     CacheIgnoreNoLastMod On
     CacheDefaultExpire 0
     CacheMaxExpire 0
     CacheIgnoreHeaders Set-Cookie
</IfModule>
```

The above configuration was provided by Ted Lawless. Restart Apache and Tomcat. Large pages should now load significantly faster for logged-out users.

You can verify http caching is occurring by looking in the directory specified as CacheRoot and seeing if files are being added. You can also use your browser's debugging tools, like Firebug or Chrome debug tools, to inspect the HTTP status code of the response for a profile page. In Chrome, enable Developer Tools (View > Developer > Developer Tools, or ⌥⌘I) and select 'Network' on the pane that appears. Cached pages will return a 304 "Not Modified" response.

9.8.5 HTTP Cache Awareness (*)

VIVO adds headers to some HTTP responses, to assist in caching profile pages

9.8.5.1 Overview

VIVO doesn't cache, but it helps to support caching.

9.8.5.2 How to enable cache awareness

What runtime properties are used to control it? Can it be controlled in developer mode?

9.8.5.3 What pages can be cached?

Only works on profile pages, and only if you are not logged in.

9.8.5.4 What do the caching headers look like?

Show a simple request with a cacheable response. Show a conditional request with a current ETag, Show a conditional request with a stale ETag.

9.8.5.5 How to configure your cache

It's up to you to insure that you don't cache something without an ETag. You should assume that all pages are stale.

9.9 Virtual Machine Templates

- Docker (see page 325)
- Vagrant (see page 325)

9.9.1 Docker

Justin Littman has created code for dockerizing VIVO 1.7 and 1.8 available on GitHub[134].

William Welling created a docker-compose.yml file (and all other necessary files), along with instructions for starting up VIVO 1.12 Docker in conjunction with a VIVO-configured Solr Docker container. It is available at the VIVO GitHub repository[135].

9.9.2 Vagrant

@Ted Lawless has created a Vagrant box to allow for quickly installing and testing the full VIVO application. The VIVO Vagrant is available on Github[136].

9.10 Moving your VIVO Instance

This page describes what you would need to do to move your VIVO instance from one machine to another.

9.10.1 Step-by-step guide

1. Make a backup of your current VIVO source directory
2. Make a backup of your current VIVO relational database
3. If different from your relational database, make a backup of your current VIVO triple store
4. Copy these backup files to your new machine
5. Create the vivo database, using the same username and password as the previous machine
6. Load the relational database from the backup
7. If you're installing everything into the same place that they were installed on the original machine, then there are no configuration changes to be made
8. Otherwise, you'll need to modify your `build.properties` in the VIVO source directory, and `runtime.properties` in the VIVO home directory, changing any paths necessary
9. If your relational database and triple store information are the same as before (same graphs, same usernames, same passwords), then there are no configuration changes to be made
10. Otherwise, you'll need to modify your `*.properties` files (see above), changing any username and password information for relational and semantic stores
11. Make sure tomcat is NOT running prior to building and installing VIVO
12. Build and install VIVO

---

[134] https://github.com/gwu-libraries/vivo-docker
[135] https://github.com/vivo-project/vivo
[136] https://github.com/lawlesst/vivo-vagrant

13. Start tomcat

And that should be it.

9.11 Regaining access to the root account

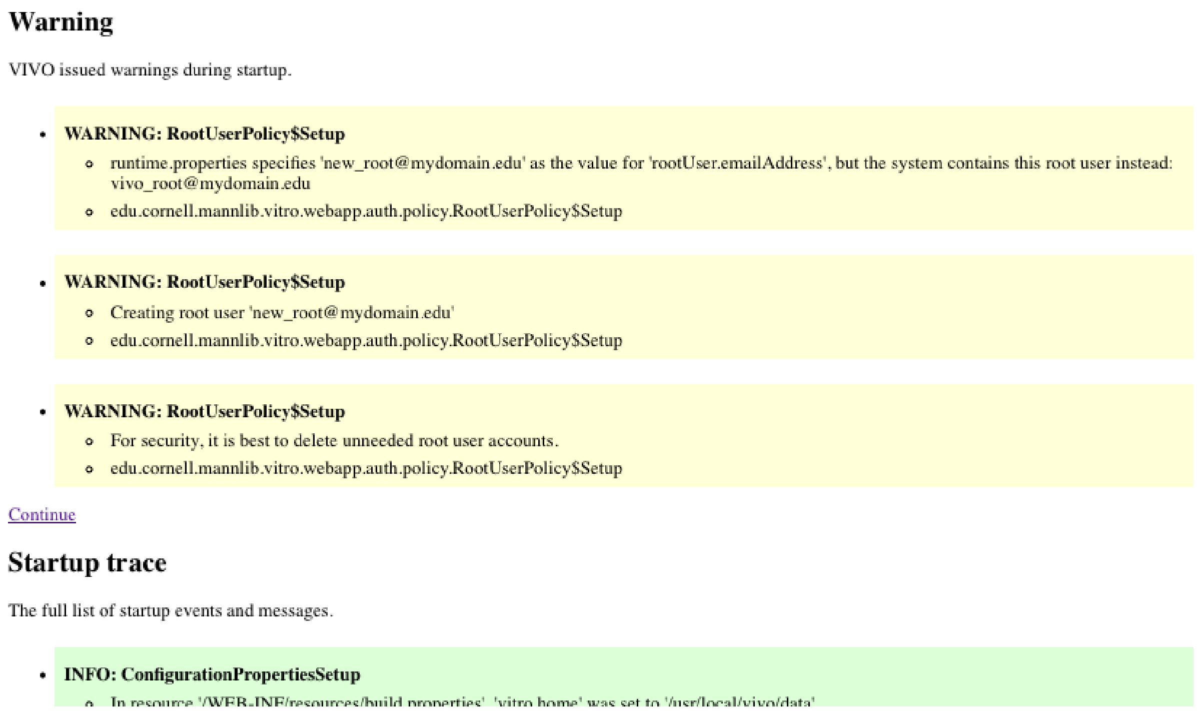
This page is intended to make access easier for VIVO developers and maintainers. An attacker cannot use these techniques to gain access to your VIVO installation. These techniques can only be used by someone who already has full access to your installation.

To gain access to the database, create a new root account.

- Modify the `runtime.properties` file to include a root account of your choosing, and restart VIVO

```
rootUser.emailAddress = new_root@mydomain.edu
```

- Open VIVO in the browser. You will see a warning screen like the following:



Warning

VIVO issued warnings during startup.

- **WARNING: RootUserPolicy$Setup**
    - runtime.properties specifies 'new_root@mydomain.edu' as the value for 'rootUser.emailAddress', but the system contains this root user instead: vivo_root@mydomain.edu
    - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup
- **WARNING: RootUserPolicy$Setup**
    - Creating root user 'new_root@mydomain.edu'
    - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup
- **WARNING: RootUserPolicy$Setup**
    - For security, it is best to delete unneeded root user accounts.
    - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup

Continue

Startup trace

The full list of startup events and messages.

- **INFO: ConfigurationPropertiesSetup**
    - In resource '/WEB-INF/resources/build.properties', 'vitro.home' was set to '/usr/local/vivo/data'

  Click `Continue` to view the VIVO home page.

- Log in using the new root account. The first-time password for your new root account will be `rootPassword`, and you will be asked to assign a new password.

You now have two root accounts, and you know the password to the new one. Use the User Accounts pages to either

- Delete the old root account,
  or
- Set a fresh password on the old root account and delete the new root account.

9.12 Altmetrics Support

9.12.1 Overview

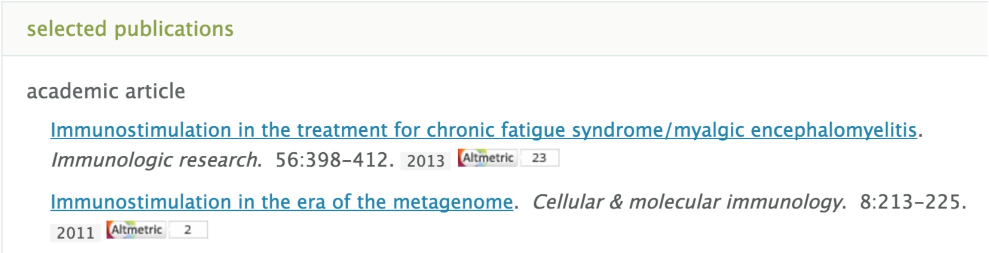
"Altmetrics" is a general term for non-traditional metrics related to scholarly works. See Wikipedia: https://en.wikipedia.org/wiki/Altmetrics

"Altmetric" is a company, a division of Digital Science, that collects altmetrics and makes them available via APIs. See http://altmetric.com

VIVO uses APIs provided by Altmetric to provide altmetrics on scholarly works. Altmetrics makes its service available without fee or license restriction.

9.12.2 Display

Scholarly works identified by DOI, PubMed ID, or ISBN have altmetric "badges" associated with them. These badges are links to altmetrics information provided by Altmetrics.



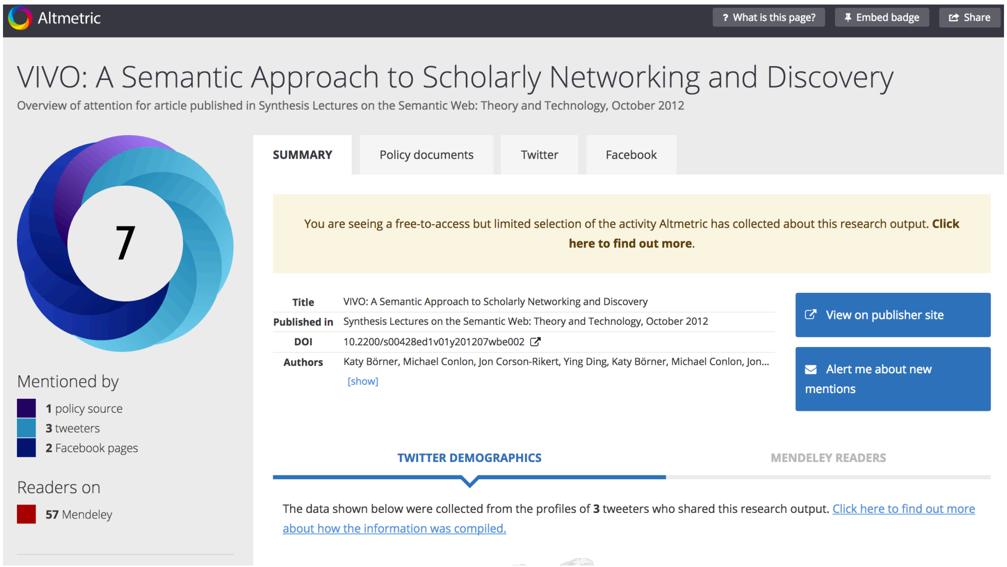
selected publications

academic article

Immunostimulation in the treatment for chronic fatigue syndrome/myalgic encephalomyelitis. *Immunologic research.* 56:398-412. 2013

Immunostimulation in the era of the metagenome. *Cellular & molecular immunology.* 8:213-225. 2011

Clicking on a badge takes you to the Altmetrics web site page for the scholarly work. For example:



VIVO: A Semantic Approach to Scholarly Networking and Discovery

Overview of attention for article published in Synthesis Lectures on the Semantic Web: Theory and Technology, October 2012

Mentioned by

- **1** policy source
- **3** tweeters
- **2** Facebook pages

Readers on

- **57** Mendeley

You are seeing a free-to-access but limited selection of the activity Altmetric has collected about this research output. **Click here to find out more**.

Title	VIVO: A Semantic Approach to Scholarly Networking and Discovery
Published in	Synthesis Lectures on the Semantic Web: Theory and Technology, October 2012
DOI	10.2200/s00428ed1v01y201207wbe002
Authors	Katy Börner, Michael Conlon, Jon Corson-Rikert, Ying Ding, Katy Börner, Michael Conlon, Jon... [show]

The data shown below were collected from the profiles of **3** tweeters who shared this research output. Click here to find out more about how the information was compiled.

9.12.3 Configuration

Six configuration parameters regulate how VIVO uses and displays altmetrics. See Configuration Reference (see page 434)

9.13 Troubleshooting

9.13.1 Having problems with your VIVO installation?

- Check your $TOMCAT DIRECTORY/logs - specifically catalina.out and vivo.all.log
- If you can't find vivo.all.log check that the data folder defined in your runtime.properties file (commonly /usr/local/vivo/home) is defined properly and is writable by Tomcat.

9.13.2 Can't find any individuals?

- First, try restarting Tomcat and go to [yourhost]/vivo/SearchIndex to see whether rebuilding the search index will fix the problem
- In the [tomcat]/logs directory, check vivo.all.log to see whether there are any error messages related to Solr
- Go to [yourhost]/vivosolr to see whether the Solr greeting page appears
    - If it does appear, then Vivo just can't reach it. Make sure that vitro.local.solr.url is set correctly in runtime.properties.
    - If you get a 403 HTTP error, then the authorization on Solr is a problem. Check your permissions.

    - If it does not appear, and you don't get a 403, then Solr did not install properly. Try cleaning the [tomcat]/webapps directory and [tomcat]/conf/Catalina/localhost directory, and rebuild VIVO using Maven
- To see your individual, go to the Site Admin page
    - click on 'Class Hierarchy'
    - navigate to the FacultyMember class link and select that link
    - on the left side of the page select the button 'show all individuals in this class'
- If an individual is found, you can select 'raw statements with this individual as subject' and you can also select 'display this individual (public)' and from there select the 'RDF' link to show the underlying RDF for the Person and some associated enitities.

9.13.3 Mail not working?

- In order for VIVO to send e-mails, it needs to have access to an SMTP server. In runtime.properties, you can set email.smtphost to the name of an SMTP server that will accept messages from your VIVO host.
- If you don't have access to an SMTP server, comment out the line for email.smtphost. VIVO will detect this, and will not attempt to send e-mails to the users. Instead, you will be required to set a password on each account as you create it, and the user will be required to change that password the first time he logs in.
- You may want to test emailing people from your server.

9.13.4 Troubleshooting Tips

9.13.4.1 Warning screen at startup

As VIVO goes through its startup process, it executes a series of "smoke tests" to try to confirm that the configuration is correct. For example, it checks to see that the home directory exists, and that VIVO has permission to write to it. It checks that VIVO can connect to the database. It checks that Solr is running, and that VIVO can connect to it.

If any of these tests fail, you will see a warning or error message when you direct your browser to VIVO. If the message is a warning (yellow), you may click the "continue" link to ignore the warning. If the message is an error (red), it is considered fatal, and VIVO will not respond to any requests.

Some of the warnings or errors may be cryptic, but they are intended to offer clues as to why your VIVO installation will not work properly.

9.13.4.2 Rebuilding the Search Index

The search index of VIVO is used not just for full text search but also for the menu pages and index pages. If the system is not displaying the individuals that you would expect to see, the search index may need to be rebuilt. To rebuild the index log in as an administrative user and request

```
http://vivo.example.edu/SearchIndex
```

This page will allow you to start a rebuild of the search index. A rebuild may take some time. The browser page will refresh every few seconds. Once the index rebuild is set up, the page will display how much time the rebuild has taken, and an estimate of how much additional time will be needed. When the indexing is completed, the page will return to its previous state.

9.13.4.3 How to Serve Linked Data

The default namespace value set during installation needs to match the domain name where you are serving your VIVO application from (VIVO web address).

Examples of VIVO web addresses and default namespace values:

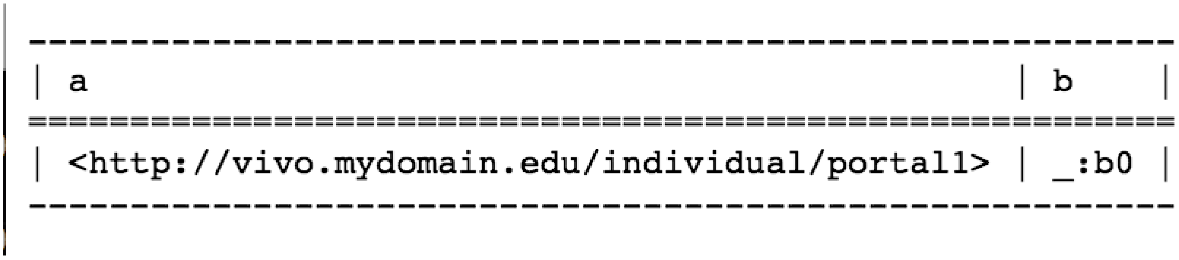
VIVO web address (url)	**Default namespace value**
http://vivo.example.edu	http://vivo.example.edu/individual
http://vivo.example.edu/vivo/	http://vivo.example.edu/vivo/individual/
http://vivoTEST.example.edu:8080/	http://vivoTEST.example.edu:8080/individual/

To check what your default namespace is currently set for:

1. Log into VIVO as an administrator, go to Site Admin -> SPARQL query.
2. Clear all of the text from the text area, enter the following query in the text area:

```
SELECT ?a ?b WHERE { ?a <http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#rootTab> ?b }
```

3. Scroll down and click “Run Query” and you should get a result like this:



```
----------------------------------------------------
| a                                        | b     |
====================================================
| <http://vivo.mydomain.edu/individual/portal1> | _:b0 |
----------------------------------------------------
```

4. To get the default namespace from the result, take everything in braces up to and including the last forward slash. In this case the default namespace is

```
http://vivo.mydomain.edu/individual/
```

5. If the default namespace does not match the domain name where your VIVO application is installed, follow the steps below:
    a. Use the “Change Namespace of Resources” option under Site Admin – Ingest Tools to set the default namespace to match your VIVO application domain name as in the above examples.
    b. Set `Vitro.defaultNamespace` in `runtime.properties` to the value for your namespace
    c. Restart Tomcat

9.13.4.4 Long URLS

If you checked your default namespace and ensured it matches the domain name where your VIVO application is installed, you may find that you still have long URLs on some people profiles.

In other words, you expect to have URLs like this: *

```
http://vivo.example.edu/individual/n5143
```

But instead, you have URLs like this:

```
http://example.edu/individual?uri=http%3A%2F%2Fvivo.example.edu%2Fsomethingl%2Fn5143
```

In this case, you have individuals with URIs that are not in your VIVO application’s default namespace. There are a couple of ways that this could have happened:

The individuals could have been created using a ingest process that did not create individuals in the default namespace.

The individuals could have been created when the system had a different default namespace.

The individuals could be from RDF data that was imported.

In general, once you have the default namespace set up correctly for your VIVO application, then all the individuals you create using the web interface will have the default namespace. You have to be careful to make sure that any individuals created by an ingest process use the default namespace.

Some individuals that are shipped with the application are not in the default namespace. For example, the countries and geographical locations are in a different namespace. Do not attempt to change the namespace of these individuals.

9.14 High Availability

9.14.1 Overview

VIVO, as delivered, is not a high availability application. Single points of failure in the application are addressed below. Some of these can be improved by approaches to deployment as noted. Others would require additional development to provide high availability deployment options.

9.14.2 Session management

VIVO code makes use of HttpSession objects. Sessions can be replicated in Tomcat and/or sticky routing to the servers.

9.14.3 Caching

VIVO does limited caching. VIVO caches some information in the visualisation stack. This is not critical to the operation of VIVO, as application servers can each build their own cache. Sticky routing, so that people get consistent graphs in a single session may be sufficient, even if each server could vary slightly in what is displayed.

9.14.4 Solr

Every server must use a single Solr cluster, rather than relying on Solr being installed alongside VIVO. Any changes being written to the index would then be shared by all instances. A shared cluster also takes care of the file system storage of Solr, which is currently maintained in the VIVO home directory.

9.14.5 Home directory

VIVO uses static configuration information, the config and rdf directories, and runtime.properties. These need to be consistent across multiple servers. That could be achieved via a shared home directory, or just multiple identical deployments.

There are three additional areas in the home directory that are of concern. The configuration triple store (tdbModels) is addressed below. Solr indexes are addressed above. The upload directory stores thumbnails for people, etc. If you allowing real-time upload of photos, this directory needs to be on a shared HA filesystem. If you are only batch ingesting thumbnails from external sources, then syncing the directory across servers could suffice. If you are simply linking to externally hosted images, the uploads folder will not be a concern.

9.14.6 Content triple store

By default, the content triple store is TDB, stored in the tdbContentModels folder in the home directory. TDB requires that you only have one JVM accessing a TDB triple store. Replication is not possible while the TDB files are open. There are two potential solutions. Through disciplined system administration you may find that the material in the content triple store can be considered static. The triple store can then be replicated across each server using a copy.

A second approach would involve storing the content triple store using SDB in an HA MySQL cluster - Configuration of Jena SDB and MySQL (see page 334).

9.14.7 Configuration triple store

The configuration triple store is TDB, stored in the tdbModels folder in the home directory. TDB requires that you only have one JVM accessing a TDB triple store. Replication is not possible while the TDB files are open. There are two potential solutions. Through disciplined system administration you may find that the material in the configuration triple store can be considered static. The triple store can then be replicated across each server using a copy. A second approach would involve storing the configuration triple store using SDB in an HA MySQL cluster. This would involve recoding relevant parts of the Vitro application, which appears to be feasible.

... (see page 332)

9.15 Replicating Ontology Changes Across Instances

9.15.1 Purpose

Suppose changes are made to the VIVO core ontology through the web interface on one VIVO instance, and these changes are needed in another instance. For example, changes are made in a development instance, tested, and approved for deployment in production. Changes may include:

- changing the display label of a core class or property
- changing the property group of a core class or property
- changing the display rank of a core class or property

The procedure below describes how such changes can be replicated between instances.

9.15.2 Procedure

In the steps below, instance #1 is the the instance that contains the changes you have made to the core ontology. Instance #2 is the instance you wish to copy the ontology changes to.

1. On instance #1, go to Site Admin > Ingest Tools > Manage Jena Models.
2. Find "http://vitro.mannlib.cornell.edu/default/asserted-tbox" and click "output model."
3. On instance #2, locate the same model and click "clear statements."
4. On instance #2, under the same model, click "load RDF data."
5. Load the file output in step 2 (N3 format).
6. Restart instance #2.

9.15.3 Best Practice

Semantic additions to the core ontology (new classes and properties) should made in a local ontology, isolated from the core ontology. Additions should be discussed on vivo-tech@googlegroups.com[137] to insure they are necessary and represent common ontological practice. Edits to the core ontology should be rare.

9.16 Jena SDB and MySQL setup

SDB vs TDB

Starting with VIVO v1.11.1, the default database of VIVO switched from Jena SDB to Jena TDB. The Jena project 'retired' SDB at the end of 2020.

VIVO can optionally use MySQL as a backing store for Jena SDB. Whilst VIVO / Jena will create the necessary tables for the triple store, first SDB must be enabled and a database (schema) and authentication details need to have been created.

First, select and enable SDB in applicationSetup.n3 by
replacing :hasContentTripleSource **:tdbContentTripleSource** ;
with :hasContentTripleSource **:sdbContentTripleSource** ;

applicationSetup.n3

```
:application
    a   vitroWebapp:application.ApplicationImpl ,
        vitroWebapp:modules.Application ;
    :hasSearchEngine              :instrumentedSearchEngineWrapper ;
    :hasSearchIndexer             :basicSearchIndexer ;
    :hasImageProcessor            :iioImageProcessor ;
    :hasFileStorage               :ptiFileStorage ;
    :hasContentTripleSource       :tdbContentTripleSource ;
    :hasConfigurationTripleSource :tdbConfigurationTripleSource ;
    :hasTBoxReasonerModule        :jfactTBoxReasonerModule .
```

Next, uncomment the block describing :sdbContentTripleSource

---

[137] mailto:vivo-tech@googlegroups.com

applicationSetup.n3

```
:sdbContentTripleSource
    a   vitroWebapp:triplesource.impl.sdb.ContentTripleSourceSDB ,
        vitroWebapp:modules.tripleSource.ContentTripleSource .
```

To create the backing database, log in to MySQL as a superuser (e.g. root)

```
$ mysql -u root -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.9 MySQL Community Server (GPL)
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE vitrodb CHARACTER SET utf8;
mysql> GRANT ALL ON vitrodb.* TO 'vitrodbUsername'@'localhost' IDENTIFIED BY
'vitrodbPassword';
mysql> FLUSH PRIVILEGES;
```

For MySQL 8+, the second command must be split into two commands like so:

```
mysql> CREATE USER 'vitrodbUsername'@'localhost' IDENTIFIED BY 'vitrodbPassword';
mysql> GRANT ALL PRIVILEGES ON vitrodb.* TO 'vitrodbUsername'@'localhost';
mysql> FLUSH PRIVILEGES;
```

Finally, you will need to edit `runtime.properties` and ensure that the VitroConnection properties are correct for your database engine. They should look something like this.

```
VitroConnection.DataSource.url = jdbc:mysql://localhost/vitrodb
VitroConnection.DataSource.username = vitrodbUsername
VitroConnection.DataSource.password = vitrodbPassword
```

9.17 How to mitigate the Log4Shell (CVE-2021-44228, CVSSv3 10.0) vulnerability

9.17.1 Log4Shell

On December 9th, 2021, a 0-day exploit in the popular Java logging library log4j was discovered that results in Remote Code Execution (RCE) by logging a certain string.

The impact of this vulnerability is quite severe. More about this issue impact (somewhere called Log4Shell) might be found at https://www.randori.com/blog/cve-2021-44228/.

9.17.2 What is affected

The VIVO core source code is **not** impacted by this vulnerability, but the Solr platform used by VIVO might be. The following versions of Solr are affected: 7.4.0 to 7.7.3, 8.0.0 to 8.11.0 (source: https://solr.apache.org/security.html#apache-solr-affected-by-apache-log4j-cve-2021-44228).

9.17.3 Mitigation

Any of the following are enough to prevent this vulnerability for Solr servers:

- Upgrade to `Solr 8.11.1` or greater (when available), which will include an updated version of the Log4J dependency.
- If you are using Solr's official docker image, no matter the version, it has already been mitigated. You may need to re-pull the image.
- Manually update the version of Log4J on your runtime classpath and restart your Solr application.
- (Linux/MacOS) Edit your `solr.in`[138]`.sh` file to include: `SOLR_OPTS="$SOLR_OPTS -Dlog4j2.formatMsgNoLookups=true"`
- (Windows) Edit your `solr.in`[139]`.cmd` file to include: `set SOLR_OPTS=%SOLR_OPTS% -Dlog4j2.formatMsgNoLookups=true`
- Follow any of the other mitgations listed at https://logging.apache.org/log4j/2.x/security.html

9.18 How to mitigate the Spring CVE-2022-22965 vulnerability

9.18.1 The CVE-2022-22965 vulnerability

On March 30th, 2022, a 0-day exploit in the popular Java framework was discovered that results in Remote Code Execution (RCE) via data binding.

---

[138] http://solr.in

[139] http://solr.in

More about this vulnerability might be found at https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement.

9.18.2 What is affected

The VIVO core source is not a Spring framework-based application, but there is dependency on spring-beans and spring-context in [VIVO]/api/pom.xml.

9.18.3 Mitigation

- Please, check the version of spring-beans and spring-context in the [VIVO]/api/pom.xml file and check whether that version is listed as affected by this vulnerability at https://mvnrepository.com/artifact/org.springframework/spring-beans
  - If in the column Vulnerabilities (https://mvnrepository.com/artifact/org.springframework/spring-beans) there is a red link to one vulnerability, please do the following:
    - replace the version tag value (https://github.com/vivo-project/VIVO/blob/main/api/pom.xml#L46) with 5.3.18, as well as the version for spring-context with 5.3.18 (https://github.com/vivo-project/VIVO/blob/main/api/pom.xml#L51)
    - redeploy VIVO (mvn clean install)

10 Reference

- APIs (see page 338)
- Application RDF Files (see page 391)
- Architecture (see page 395)
- Configuration Reference (see page 434)
- Directories and Files (see page 440)
- Freemarker Template Variables and Directives (see page 443)
- Graph Reference (see page 443)
- Ontology Reference (see page 446)
- Resource Links (see page 476)
- Rich export SPARQL queries (see page 478)
- URL Reference (see page 501)
- Utilities Reference (see page 502)

10.1 Overview

This section contains reference material for the VIVO and Vitro systems. These materials take the form of glossaries and lists. They are not intended in the form of instructional materials. For processes used to support VIVO and Vitro, see the System Administration (see page 294) section and the introductory sections in particular for processes and instructional material.

10.2 APIs

- Data Distribution API (see page 339)
- Direct2Experts API (see page 345)
- Linked Open Data - requests and responses (see page 346)
- ListRDF API (see page 355)
- Reconciliation API (see page 359)
- Search indexing service (see page 362)
- SPARQL Query API (see page 366)
- SPARQL Update API (see page 372)
- Triple Pattern Fragments (see page 380)

The VIVO APIs are HTTP end-points that can be used to read or write data, or to manage VIVO's operation. Other than Triple Pattern Fragments (see page 380), they have no user interface, and are intended to be called by external applications that are cooperating with VIVO.

The end-points include:

Public Services	- **available without restriction** - **provide filtered results, allowing restrictions on data**
Linked Open Data (see page 346)	Information about an individual, its types, its data values, incoming and outgoing links.
ListRDF (see page 355)	Lists of individuals that belong to a particular class in the ontology. For example, a list of all People, or all Organizations.
Triple Pattern Fragments (see page 380)	Lists of triples that match triple patterns. Can be used to retrieve all triples.
Data Distribution API (see page 339)	Create custom end points to provide data in multiple formats
Direct2Experts API (see page 345)	Provides data in response to a request from Direct2Experts.

Access Controlled Services	- **require account credentials on each request** - **credentials are for an internal VIVO with sufficient authorization** - **results are not filtered, and may return data that should be kept private**
SPARQL Query API (see page 366)	Submit a SPARQL query to get information from VIVO. Supports `SELECT`, `ASK`, `CONSTRUCT`, and `DESCRIBE` query types.
SPARQL Update API (see page 372)	Submit a SPARQL query to `INSERT` new triples or `DELETE` existing triples. Also, `LOAD` triples from a web-accessible file.
Search Indexing API (see page 362)	Submit a list of URIs that may have stale data in the search index. The search data for each of these URIs will be rebuilt.

10.2.1 Data Distribution API

10.2.1.1 Overview

The Data Distribution API is used to create data feeds from your VIVO site by editing a configuration file.

Use data feeds to:

- provide content to other sites
- service AJAX requests from your own VIVO pages
- provide a more responsive user interface on your VIVO site
- drive visualizations of your VIVO data

Get more use of the data in your VIVO site

- without opening your site to expensive queries
- without digging into the internals of VIVO
- without writing any Java code

You will likely need to know SPARQL, Turtle syntax for RDF, and the structure of your data in VIVO.

For more on the motivation and design philosophy behind the Data Distribution API, see Data Distribution motivation[140]

10.2.1.2 Hello, World

As a first example, we create an API that can be called to return the text "Hello, World" – not very useful, but shows the basic mechanism for creating and using an API based on a configuration file.

First, create a Turtle format file in `<vivo-home>/rdf/display/everytime` You may name the file anything you like. All example files are in Turtle format. Turtle files typically have a `.ttl` file type. Your file should contain the following:

```
@prefix : <http://vitro.mannlib.cornell.edu/ns/vitro/ApplicationSetup#> .

:data_distributor_hello
    a
<java:edu.cornell.library.scholars.webapp.controller.api.distribute.DataDistributor>
,

<java:edu.cornell.library.scholars.webapp.controller.api.distribute.examples.HelloDis
tributor> ;
    :actionName "hello" .
```

Second, restart Tomcat. VIVO will reread its everytime content. The assertions in the Turtle file will be added to VIVO's content triple store. The API is now ready for use.

Third, visit the URL: `http://mydomain.edu/api/datarequest/hello`

You should see:

```
Hello, World
```

in your browser.

---

[140] https://cul-it.github.io/vivo-data-distribution-api/motivation.html

10.2.1.2.1 Parameters

The HelloDistributor supports a name parameter which can be specified in the call:

Visit `http://mydomain.edu/api/datarequest/hello?name=Bob`

You should see:

```
Hello, Bob
```

in your browser.

The Data Distribution API supports parameterized requests, making it ideal for getting data about a particular person, a particular publication, all the people in a particular department, and so on.

10.2.1.2.2 How to Read the Configuration File

The configuration file begins with a prefix. Prefixes provide shortcuts for URIs used in RDF. See the Turtle reference for additional background and examples.

Following are two assertions of type for a new entity with the URI `:data_distributor_hello`. You can use any URI for your data distributor entity. You will need to use unique URI for each data distributor entity you create, so a naming convention (see below) may be helpful.

The type assertions are required and declare that the entity is a data distributor, and is a HelloDistributor. Subsequent examples will use other types of data distributors.

The third assertion says that the entity has the actionName "hello". This is the API's key for someone calling the API. The entity will be called when the URL http://mydomain.edu/api/datarequest/hello is visited. The "/api/datarequest" part of the URL is fixed and indicates you are making a request of the VIVO Data Distribution API. The final part of the URL is the action name specified in the configuration file.

10.2.1.3 Managing Configuration Files

Your configuration files will live in `<vivo-home>/rdf/display/everytime`

You may wish to use a naming convention to identify Data Distribution API configuration files.

You will want to preserve your configuration files through upgrades and builds of VIVO. See Preserving Customizations During Build (see page 51)

10.2.1.4 References

1. VIVO Data Distribution API on GitHub https://vivo-community.github.io/vivo-data-distribution-api/
2. Turtle - Terse RDF Triple Language https://www.w3.org/TeamSubmission/turtle/
3. Learning SPARQL http://learningsparql.com
4. VIVO Ontology Reference (see page 446)

10.2.1.5 Data Distribution Predicates

The Data Distribution API uses RDF statements of the form subject predicate object to specify the configuration for an action. Each distributor has a type (specified by "a" see below), an action name, and may use additional predicates to indicate relations to other entities. Multiple distributors may be used in the a single configuration to drill down, iterate, and/or assemble graphs.

The table includes all predicates, but only certain predicates can be used with specific data distributor. Check the documentation for the distributor you are using to determine which predicates apply to that distributor.

Predicate	**Type**	**Usage**
a	O	Specify the type of data distributor
actionName	D	Associates the instance with an HTTP request.
child	O	The “wrapped” Data Distributor
childGraphBuilder	O	The “decorated” GraphBuilder instance(s), which will produce the RDF graph.
constructQuery	D	The SPARQL CONSTRUCT query
contentType	D	The MIME type to be sent in the HTTP response header.
drillDownQuery	D	Discovers the values that will be passed to the child GraphBuilder instance(s).
emptyResponse	D	A string to be served as an “empty data set”, if the file is not found.
filepathTemplate	D	A template for constructing the file path from the selection value. If the constructed path is relative, it is relative to the Vitro home directory.
graphBuilder	O	Creates an internal RDF graph
literalBinding	D	The name of a request parameter whose value should be bound in the query as a plain literal.

Predicate	Type	Usage
parameterName	D	The request parameter name. Used in various contexts by the distributors. See the examples.
parameterPattern	D	A regular expression to extract the file selector from the parameter value.
parameterValue	D	A that will be added to the named parameter, each value will be used in a run of childGraphBuilder
path	D	The location of a file to be served by the FileDistributor. If a relative path, it is relative to the Vitro home directory.
query	D	The SPARQL SELECT query.
script	D	A string of JavaScript to be interpreted and executed. It must contain a function named transform which accepts a String as argument and returns a String as result.
supportingScript	D	The path to a JavaScript file in the webapp. The path must be as specified for ServletContext.getResource()[141]. That is, it must begin with a '/' and is interpreted as relative to the context root of the webapp.
topLevelGraphBuilder	O	The source of the top-level graph, against which the drillDownQuery is run.
uriBinding	D	The name of a request parameter whose value should be bound in the query as a URI.

10.2.1.6 Data Distributors

The Data Distribution API supports the following distributors. These can be used in combination in a configuration file to define an action, parameters, and response.

---

[141] https://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html#getResource-java.lang.String-

Name	Description
HelloDistributor	The Hello World Demonstration
FileDistributor	Sends the contents of a file as an HTTP response
SelectingFileDistributor	Serves the contents of a file. Selects the file by extracting a value from the HTTP request, and using it to construct the file path.
SelectFromContentDistributor	Executes a SPARQL SELECT query against Vitro's content triple-store, and returns the results in JSON format: application/sparql-results+json.
RdfGraphDistributor	Executes one or more GraphBuilder[142] instances. Merges the results into an internal RDF graph, and returns that graph in Turtle format: text/turtle.
SelectFromGraphDistributor	Executes a SPARQL SELECT query against an internal RDF graph. The graph is created by merging the outputs of one or more GraphBuilder instances. The results are returned in JSON format: application/sparql-results+json.
EmptyGraphBuilder	Creates an RDF graph containing no triples. Used for tests, examples, or placeholders.
ConstructQueryGraphBuilder	Executes SPARQL CONSTRUCT query(s) against Vitro's content triple-store, and returns a model that contains the (merged) results.
IteratingGraphBuilder	A GraphBuilder decorator that runs one or more "child" builders multiple times, each time providing a different value for a specified request parameter. The results of all queries are merged into a local RDF graph.
DrillDownGraphBuilder	A GraphBuilder decorator that runs one or more "child" builders multiple times, each time providing a different value for a specified request parameter. The results of all queries are merged into a local RDF graph.
JavaScriptTransformDistributor	A wrapper that uses a JavaScript function to transform the output of the wrapped distributor.

---

[142] https://cul-it.github.io/vivo-data-distribution-api/catalog.html#ardfgraphbuilder_package

10.2.1.6.1 Adding Distributors

Additional distributors can be added. See https://vivo-community.github.io/vivo-data-distribution-api

10.2.2 Direct2Experts API

10.2.2.1 Overview

Direct2Experts is a cross-site search capability for identifying expertise as expressed by research networking systems such as VIVO, PURE, and Harvard Profiles. Every VIVO site version 1.10 and later has Direct2Experts capability built-in.

To see Direct2Experts in action, visit http://direct2experts.org

10.2.2.2 The Direct2Experts Endpoints

VIVO uses two endpoints to provide Direct2Experts services:

1. The bootstrap endpoint. For VIVO, this is /FS.xml. A small XML file is returned describing your VIVO endpoint. Your application name, set in application-settings.xml is returned by FS.xml
2. The aggregated query endpoint. For VIVO, this is /ctsasearch.

10.2.2.3 Participating in Direct2Experts

Once your VIVO is in production, email your bootstrap endpoint URL to Griffin Weber (weber at hms dot harvard dot edu). In the email please include the main URL of your VIVO website, indicate you use VIVO, so you will be added to the Participants[143] page.

10.2.2.4 References

1. Weber GM, Barnett W, Conlon M, Eichmann D, Kibbe W, Falk-Krzesinski H, Halaas M, Johnson L, Meeks E, Mitchell D, Schleyer T, Stallings S, Warden M, Kahlon M; Direct2Experts Collaboration. Direct2Experts: a pilot national network to demonstrate interoperability among research-networking platforms. J Am Med Inform Assoc. 2011 Dec;18 Suppl 1:i157-60. doi: 10.1136/amiajnl-2011-000200. Epub 2011 Oct 28.
2. Direct2Experts reference implementation on GFitHub. https://github.com/eichmann/direct2experts-reference-implementation
3. Code here: https://github.com/OIT-ADS-Web/vivo/pull/62/files
4. Direct2Experts Technical Documentation: http://direct2experts.org/join

---

[143] http://direct2experts.org/?pg=participants

10.2.3 Linked Open Data - requests and responses

10.2.3.1 Overview

Linked Open Data is one of the fundamental concepts of the Semantic Web. It consists of asking a server for the RDF relating to an individual. If the response includes object properties that link to other individuals, those individuals can be queried also. For more information on Linked Open Data, see Concept: Linked Data (see page 346).

VIVO accepts standard requests for Linked Open Data and some non-standard ones. The contents of the response are in accordance with those suggested by the in their tutorial How to Publish Linked Data on the Web[144].

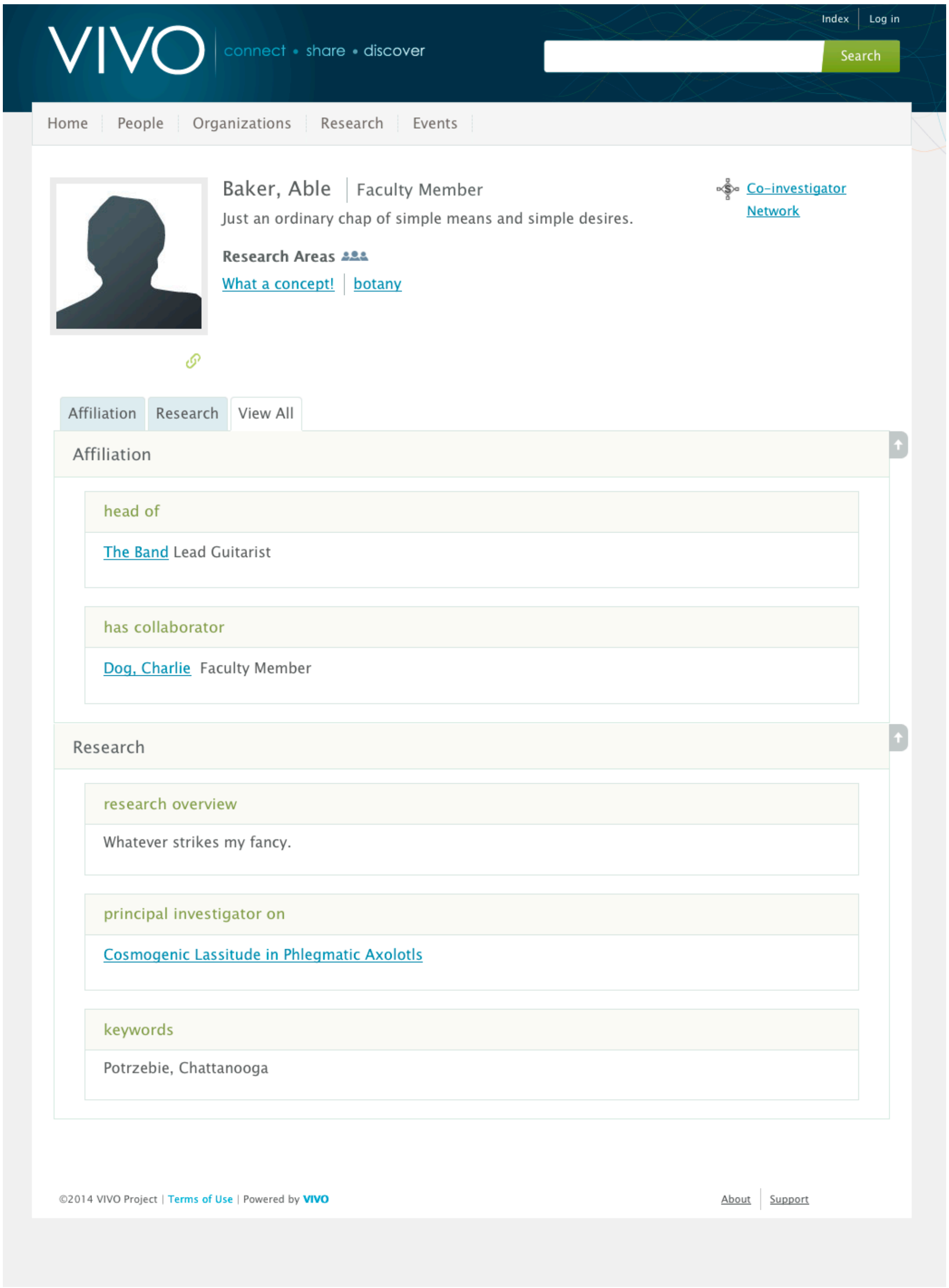
VIVO will provide Linked Open Data in several formats. The semantic content remains the same; only the syntax differs among formats.

10.2.3.1.1 An example

The examples on this page are based on a fictitious individual named "Able Baker", with a URI of `http://vivo.mydomain.edu/individual/n3639`. To keep the examples simple, this person has just a few items in his VIVO profile. His profile page looks like this:

---

[144] http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/


VIVO
connect • share • discover
Index
Log in
Search
Home
People
Organizations
Research
Events
Co-investigator Network
Affiliation
Research
View All
About
Support

Baker, Able | Faculty Member

Just an ordinary chap of simple means and simple desires.

Research Areas

What a concept! | botany

Affiliation

head of

The Band Lead Guitarist

has collaborator

Dog, Charlie Faculty Member

Research

research overview

Whatever strikes my fancy.

principal investigator on

Cosmogenic Lassitude in Phlegmatic Axolotls

keywords

Potrzebie, Chattanooga

©2014 VIVO Project | Terms of Use | Powered by VIVO

10.2.3.2 Requesting Linked Open Data from VIVO

10.2.3.2.1 Available formats

VIVO will serve Linked Open Data in these formats:

- RDF/XML[145]
- N3[146]
- Turtle[147]
- JSON-LD[148]

Specifications for each of the formats are provided by the World Wide Web Consortium (W3C).

10.2.3.2.2 Types of requests

The standard way of requesting Linked Open data is an HTTP request to the URI of the individual in question, with the `Accept` header on the request indicating the desired format. If there is no `Accept` header, it is assumed to be `text/html`, and the standard profile page is returned.

URL	Accept header	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639`	`application/rdf+xml`	RDF/XML	`application/rdf+xml`
`http://vivo.mydomain.edu/individual/n3639`	`text/n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639`	`text/turtle`	Turtle	`text/turtle`
`http://vivo.mydomain.edu/individual/n3639`	`application/json`	JSON-LD	`application/json`

The different responses may also be explicitly requested by URL. In fact, the requests listed above will simply redirect the browser to these URLs:

URL	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639/n3639.rdf`	RDF/XML	`application/rdf+xml`

---

[145] http://www.w3.org/TR/REC-rdf-syntax/
[146] http://www.w3.org/TeamSubmission/n3/
[147] http://www.w3.org/TeamSubmission/turtle/
[148] http://www.w3.org/TR/json-ld/

URL	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639/n3639.n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639/n3639.ttl`	Turtle	`text/turtle`
`http://vivo.mydomain.edu/individual/n3639/n3639.jsonld`	JSON-LD	`application/json`

Finally, VIVO allows you to request Linked Open Data in a way that is not specified by the standard. You can make an HTTP GET request to the URI of the individual, and include a `format` parameter that specifies the format of the response.

URL	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639?format=rdfxml`	RDF/XML	`application/rdf+xml`
`http://vivo.mydomain.edu/individual/n3639?format=n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639?format=ttl`	Turtle	`text/turtle`
`http://vivo.mydomain.edu/individual/n3639?format=jsonld`	JSON-LD	`application/json`

10.2.3.3 What is included in the response?

When you get request the public RDF about an individual in VIVO, the result is a set of RDF statements, or triples. These triples state:

- The data properties of the individual.
- The object properties that relate this individual to other individuals.
- The object properties of other individuals that relate to this individual
- The labels and types of these related individuals.
- Some triples that describe the RDF document itself.

This statement over-simplifies slightly. In VIVO, object properties and data properties can be public, or restricted to some extent. The RDF for an individual will contain only public properties.

10.2.3.3.1 An example response

Here is the RDF produced for the example, in N3 format.

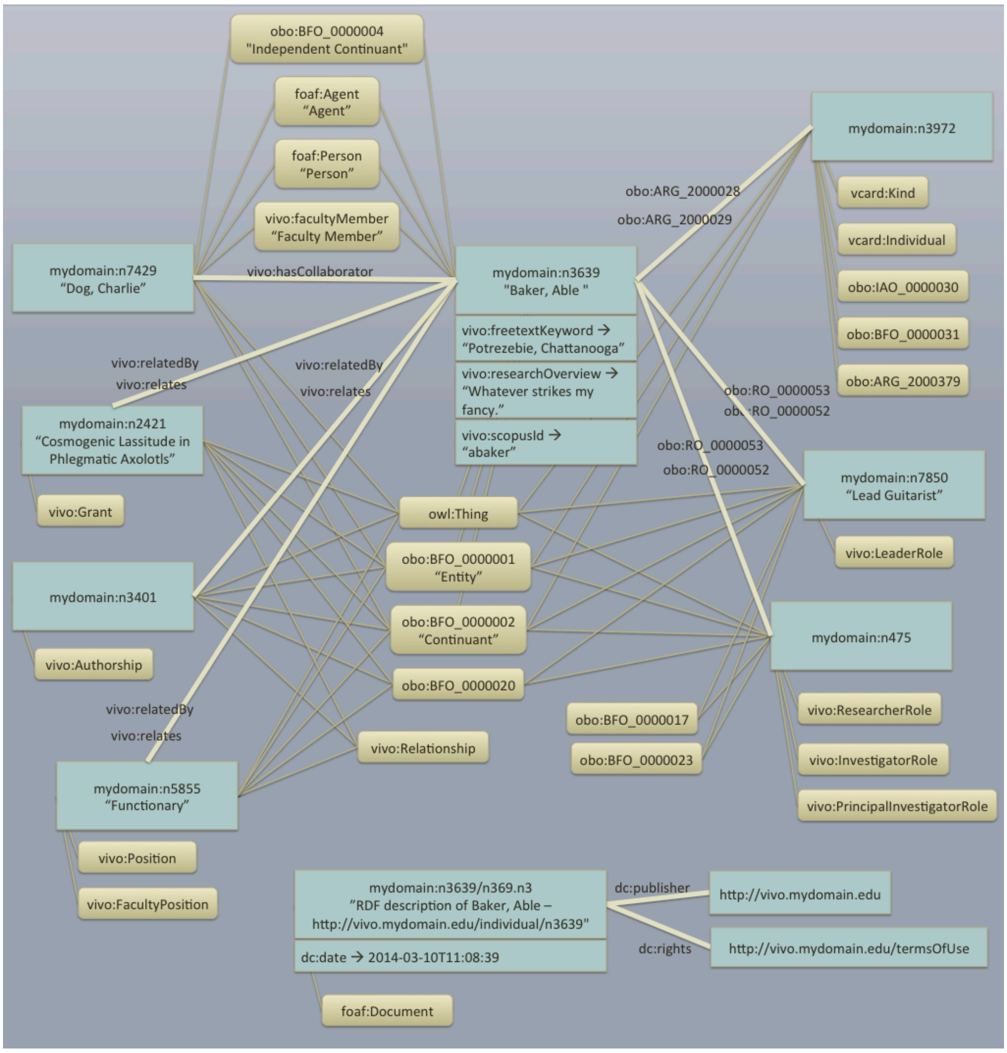
```
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix vcard:   <http://www.w3.org/2006/vcard/ns#> .
@prefix obo:     <http://purl.obolibrary.org/obo/> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix vivo:    <http://vivoweb.org/ontology/core#> .

<http://vivo.mydomain.edu/individual/n3639>
      a       vivo:FacultyMember ,
              foaf:Person ,
              owl:Thing ,
              foaf:Agent ,
              obo:BFO_0000002 ,
              obo:BFO_0000001 ,
              obo:BFO_0000004 ;
      rdfs:label "Baker, Able "^^xsd:string ;
      obo:ARG_2000028 <http://vivo.mydomain.edu/individual/n3972> ;
      obo:RO_0000053 <http://vivo.mydomain.edu/individual/n475> ,
                     <http://vivo.mydomain.edu/individual/n7850> ;
      vitro:mostSpecificType
              vivo:FacultyMember ;
      vivo:freetextKeyword
              "Potrezebie, Chattanooga" ;
      vivo:hasCollaborator
              <http://vivo.mydomain.edu/individual/n7429> ;
      vivo:relatedBy <http://vivo.mydomain.edu/individual/n3401> ,
                     <http://vivo.mydomain.edu/individual/n5855> ,
                     <http://vivo.mydomain.edu/individual/n2421> ;
      vivo:researchOverview
              "Whatever strikes my fancy." ;
      vivo:scopusId "abaker" .

<http://vivo.mydomain.edu/individual/n3972>
      a       vcard:Kind ,
              obo:BFO_0000031 ,
              owl:Thing ,
              obo:ARG_2000379 ,
              obo:IAO_0000030 ,
              obo:BFO_0000002 ,
              obo:BFO_0000001 ,
              vcard:Individual ;
```

```
        obo:ARG_2000029 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n475>
        a       owl:Thing ,
                obo:BFO_0000023 ,
                vivo:InvestigatorRole ,
                obo:BFO_0000002 ,
                obo:BFO_0000017 ,
                vivo:PrincipalInvestigatorRole ,
                obo:BFO_0000020 ,
                obo:BFO_0000001 ,
                vivo:ResearcherRole ;
        obo:RO_0000052 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n7850>
        a       owl:Thing ,
                obo:BFO_0000023 ,
                obo:BFO_0000017 ,
                obo:BFO_0000002 ,
                obo:BFO_0000020 ,
                obo:BFO_0000001 ,
                vivo:LeaderRole ;
        rdfs:label "Lead Guitarist"^^xsd:string ;
        obo:RO_0000052 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n7429>
        a       foaf:Person ,
                vivo:FacultyMember ,
                foaf:Agent ,
                owl:Thing ,
                obo:BFO_0000002 ,
                obo:BFO_0000001 ,
                obo:BFO_0000004 ;
        rdfs:label "Dog, Charlie" .

<http://vivo.mydomain.edu/individual/n3401>
        a       owl:Thing ,
                vivo:Relationship ,
                obo:BFO_0000002 ,
                obo:BFO_0000020 ,
                obo:BFO_0000001 ,
                vivo:Authorship ;
        vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n5855>
        a       vivo:FacultyPosition ,
                owl:Thing ,
                vivo:Relationship ,
                obo:BFO_0000002 ,
                obo:BFO_0000020 ,
                obo:BFO_0000001 ,
                vivo:Position ;
```

```
      rdfs:label "Functionary"^^xsd:string ;
      vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n2421>
      a       owl:Thing ,
              vivo:Relationship ,
              obo:BFO_0000002 ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:Grant ;
      rdfs:label "Cosmogenic Lassitude in Phlegmatic Axolotls" ;
      vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

obo:BFO_0000001
      a       owl:Class ;
      rdfs:label "Entity" .

obo:BFO_0000002
      a       owl:Class ;
      rdfs:label "Continuant" .

obo:BFO_0000004
      a       owl:Class ;
      rdfs:label "Independent Continuant"@en-US .

vivo:FacultyMember
      a       owl:Class ;
      rdfs:label "Faculty Member"@en-US .

foaf:Person
      a       owl:Class ;
      rdfs:label "Person"@en-US .

foaf:Agent
      a       owl:Class ;
      rdfs:label "Agent"@en-US .

owl:Thing
      a       owl:Class .

<http://vivo.mydomain.edu/individual/n3639/n3639.n3>
      a       foaf:Document ;
      rdfs:label "RDF description of Baker, Able  - http://vivo.mydomain.edu/
individual/n3639" ;
      <http://purl.org/dc/elements/1.1/date> "2014-03-10T11:08:39"^^xsd:dateTime ;
      <http://purl.org/dc/elements/1.1/publisher> <http://vivo.mydomain.edu> ;
      <http://purl.org/dc/elements/1.1/rights> <http://vivo.mydomain.edu/
termsOfUse> .
```

10.2.3.3.2 A graphic summary

The RDF can be expressed graphically like this:


obo:BFO_0000004
"Independent Continuant"
foaf:Agent
"Agent"
foaf:Person
"Person"
vivo:facultyMember
"Faculty Member"
mydomain:n3972
vcard:Kind
vcard:Individual
obo:IAO_0000030
obo:BFO_0000031
obo:ARG_2000379
obo:ARG_2000028
obo:ARG_2000029
mydomain:n7429
"Dog, Charlie"
vivo:hasCollaborator
mydomain:n3639
"Baker, Able "
vivo:freetextKeyword → "Potrezebie, Chattanooga"
vivo:researchOverview → "Whatever strikes my fancy."
vivo:scopusId → "abaker"
vivo:relatedBy
vivo:relates
vivo:relatedBy
vivo:relates
obo:RO_0000053
obo:RO_0000052
obo:RO_0000053
obo:RO_0000052
mydomain:n2421
"Cosmogenic Lassitude in Phlegmatic Axolotls"
vivo:Grant
owl:Thing
mydomain:n7850
"Lead Guitarist"
vivo:LeaderRole
obo:BFO_0000001
"Entity"
mydomain:n3401
obo:BFO_0000002
"Continuant"
mydomain:n475
vivo:Authorship
obo:BFO_0000020
vivo:relatedBy
vivo:relates
obo:BFO_0000017
vivo:ResearcherRole
vivo:Relationship
obo:BFO_0000023
vivo:InvestigatorRole
mydomain:n5855
"Functionary"
vivo:PrincipalInvestigatorRole
vivo:Position
vivo:FacultyPosition
mydomain:n3639/n369.n3
"RDF description of Baker, Able – http://vivo.mydomain.edu/individual/n3639"
dc:publisher
http://vivo.mydomain.edu
dc:date → 2014-03-10T11:08:39
dc:rights
http://vivo.mydomain.edu/termsOfUse
foaf:Document

10.2.3.4 Restricting properties

10.2.3.4.1 Editing the property

You can exclude a property from Linked Open Data, or include it, by editing the property within VIVO. Perhaps the easiest way to edit a property is to log in as a VIVO administrator, navigate to an individual's profile page, and turn on the verbose display:

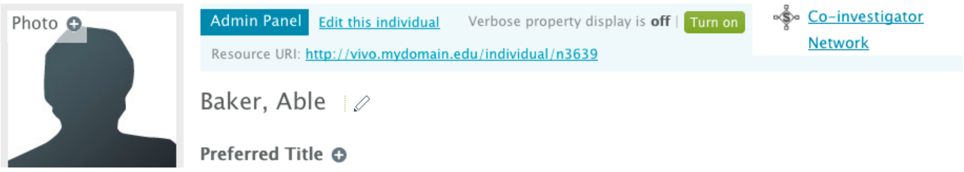
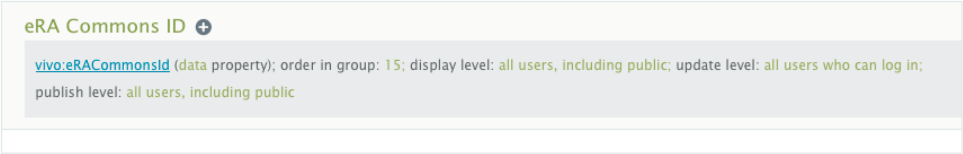


Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on
Co-investigator Network
Resource URI: http://vivo.mydomain.edu/individual/n3639
Baker, Able
Preferred Title

Once the verbose display is turned on, scroll through the profile page to find the property you are interested in. You can see what it's current restriction levels are for display, update, and publishing. You also have a link to the control panel for that property:


eRA Commons ID

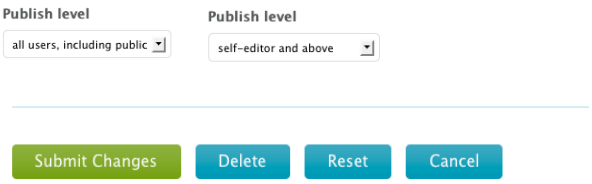
vivo:eRACommonsId (data property); order in group: 15; display level: all users, including public; update level: all users who can log in; publish level: all users, including public

Note that all Linked Open Data requests are treated as public, so any setting other than `all users, including public` will exclude the property.

Navigate to the control panel for the property, and then to the editing form for the property.

Edit this Data Property

Set the `Publish level` as you like, and submit the changes.


Publish level: all users, including public

Publish level: self-editor and above

Submit Changes
Delete
Reset
Cancel

10.2.3.4.2 Setting triples in the display model

Properties in VIVO can be restricted from Linked Open Data, by attaching the `vitro:hiddenFromPublishBelowRoleLevelAnnot` annotation to the property.

For example, this triple in VIVO's display model would mean that the `eRACommonsId` property would not be published in Linked Open Data

```
<http://vivoweb.org/ontology/core#eRACommonsId>
      <http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#hiddenFromPublishBelowRoleLevelAnnot>
              <http://vitro.mannlib.cornell.edu/ns/vitro/role#nobody> .
```

Note, however, that the standard VIVO distribution includes this triple in the display model:

```
<http://vivoweb.org/ontology/core#:eRACommonsId>
```

```
      <http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#:hiddenFromPublishBelowRoleLevelAnnot>
              <http://vitro.mannlib.cornell.edu/ns/vitro/role#public> .
```

You would need to remove this triple in order for the more restrictive triple to take effect.

10.2.3.4.3 An exception to the restrictions

VIVO uses the same permissions model to restrict Linked Open Data that it uses to restrict displays or updates. So if you are logged in to VIVO as the root user, and you request Linked Open Data, no restrictions would be applied.

This is consistent with VIVO's authorization model.

An external application could take advantage of this fact to obtain full RDF about individuals. Since there is no authorization parameter on the Linked Open Data request, the client application would need to begin by logging in to VIVO as an administrator, and then retain the session cookie to submit with subsequent requests.

10.2.3.5 Error handling

If you ask for Linked Open Data for a non-existent individual, regardless of the form you use, VIVO will return a response code of `404 not found`.

If you ask for an unsupported format, either in the `Accept` header or the `format` parameter, VIVO will treat your request as a request for HTML, and will return the standard profile page for the individual. The response code will be `200 OK`.

10.2.4 ListRDF API

10.2.4.1 Overview

10.2.4.1.1 Purpose

Permits external applications to obtain a list of all Individuals in VIVO that belong to a specified class. For example, a list of all Persons, or a list of all Organizations.

This API complements the Linked Open Data API. The Linked Open Data standard describes a way to get data about any Individual, but it does not provide a way to get a list of Individuals to begin with.

10.2.4.1.2 Filtered results

The results of this query is filtered by the same VIVO policies that control Linked Open Data. Individuals may be omitted from the results, if those policies restrict access to those Individuals.

10.2.4.1.3 Use Cases

10.2.4.1.3.1 Harvesting data from VIVO

Data in VIVO is available to other applications via Linked Open Data - requests and responses (see page 346). A list of Individuals from this API may provide a starting point for such applications.

10.2.4.1.3.2 Multi-site search index

If an external application chooses to build a compendium from several VIVO sites, it will need to know what Individuals are present in each site.

10.2.4.2 Specification

10.2.4.2.1 URL

`[vivo]/listrdf`

Examples:

```
http://vivo.cornell.edu/listrdf
```

```
http://localhost:8080/vivo/listrdf
```

10.2.4.2.2 HTTP Method

The API supports HTTP GET or POST calls.

10.2.4.2.3 Parameters

name	value
`vclass`	the URI of the class to be listed.

10.2.4.2.4 Response Codes

Code	Reason
`200 OK`	SPARQL query was successful.
`400 Bad Request`	HTTP request did not include a `vclass` parameter.
`406 Not Acceptable`	The Accept header does not include any available content types.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.2.4.2.5 Content of the response

The response will contain RDF triples. Each triple asserts that an Individual is an instance of the requested class.

10.2.4.2.6 Available content types

The request may include an `Accept` header, to specify the preferred content type of the response. If no `Accept` header is provided, the preferred content type is assumed to be `text/plain`.

MIME type in the Accept header	Response format	Format description
`text/plain`	N-Triples	http://www.w3.org/2001/sw/RDFCore/ntriples/
`application/rdf+xml`	RDF/XML	http://www.w3.org/TR/rdf-syntax-grammar/
`text/n3`	N3	http://www.w3.org/TeamSubmission/n3/
`text/turtle`	Turtle	http://www.w3.org/TeamSubmission/turtle/

MIME type in the Accept header	Response format	Format description
`application/json`	JSON-LD	http://www.w3.org/TR/json-ld/

10.2.4.3 Examples

These examples use the UNIX `curl` command to issue queries to the API.

10.2.4.3.1 Continents as N-Triples example

This example requests a list of `vivo:Continent` Individuals, in N-triples format.

```
curl -i -d 'vclass=http://vivoweb.org/ontology/core#Continent' -H 'Accept:text/plain'
 'http://localhost:8080/vivo/listrdf'
```

The response looks like this:

```
<http://aims.fao.org/aos/geopolitical.owl#Africa> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Europe> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Antarctica> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#northern_America> <http://www.w3.org/
1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#South_America> <http://www.w3.org/
1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Australia_and_New_Zealand> <http://
www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/
core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Asia> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
```

10.2.4.3.2 Faculty Members as JSON-LD example

This example requests a list of `vivo:FacultyMember` Individuals, in JSON.

```
curl -i -d 'vclass=http://vivoweb.org/ontology/core#FacultyMember' -H
'Accept:application/json' 'http://localhost:8080/vivo/listrdf'
```

The response (for a very small VIVO) looks like this:

```
[{"@id":"http://vivo.mydomain.edu/individual/n4295","@type":["http://vivoweb.org/ontology/core#FacultyMember"]},{"@id":"http://vivo.mydomain.edu/individual/n5056","@type":["http://vivoweb.org/ontology/core#FacultyMember"]},{"@id":"http://vivo.mydomain.edu/individual/n7630","@type":["http://vivoweb.org/ontology/core#FacultyMember"]},{"@id":"http://vivoweb.org/ontology/core#FacultyMember"}]
```

10.2.5 Reconciliation API

10.2.5.1 Purpose

The Reconciliation API or Reconciliation Endpoint provides the ability to reconcile data in OpenRefine[149] with that from a VIVO and prepare it for import into VIVO.

OpenRefine describes itself as a "powerful tool for working with messy data: cleaning it; transforming it from one format into another; and extending it with web services and external data". One use case of OpenRefine is to match tabular data with data in a VIVO, export it as RDF and then import it into a VIVO.

10.2.5.2 Specification

10.2.5.2.1 URL:

```
[vivo]/reconcile
```

10.2.5.2.2 Example:

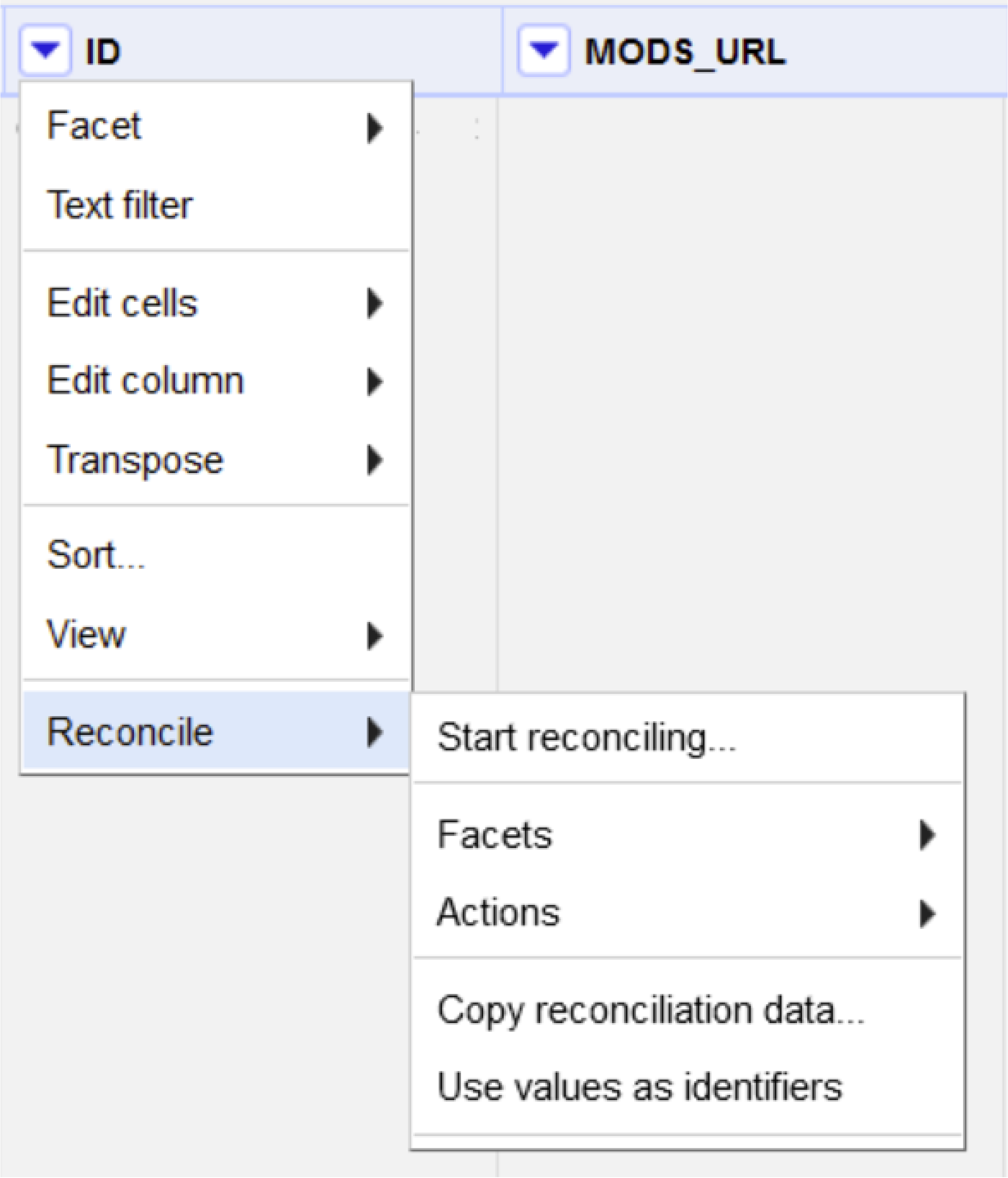
```
https://openvivo.org/reconcile
```

---

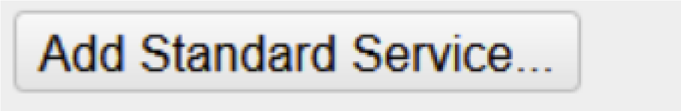
[149] https://openrefine.org/

10.2.5.3 Usage with OpenRefine

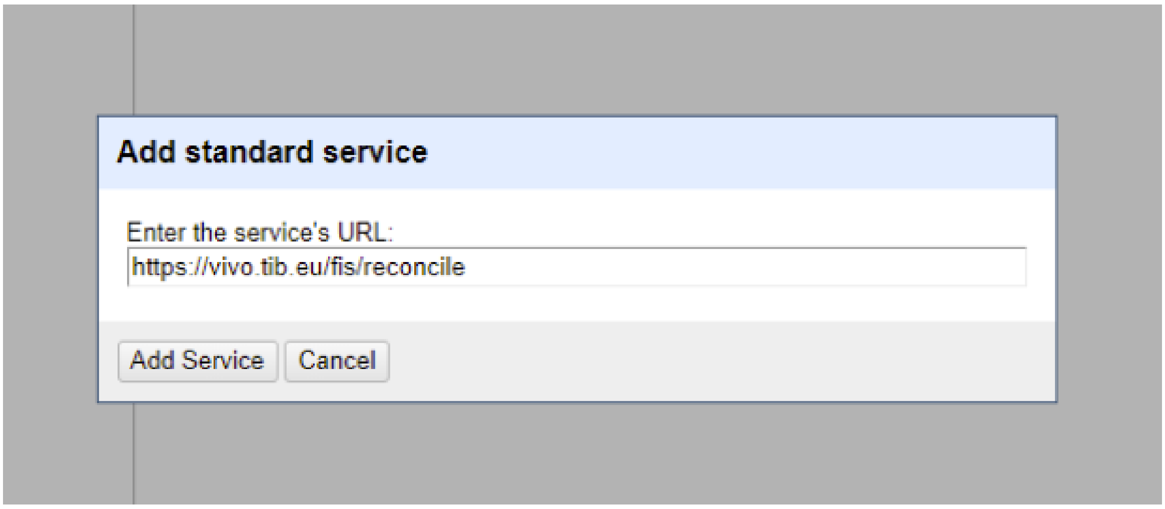
1. Click on “Start reconciling” in OpenRefine


ID
MODS_URL
Facet
Text filter
Edit cells
Edit column
Transpose
Sort...
View
Reconcile
Start reconciling...
Facets
Actions
Copy reconciliation data...
Use values as identifiers

2. Click on “Add Standard Service”



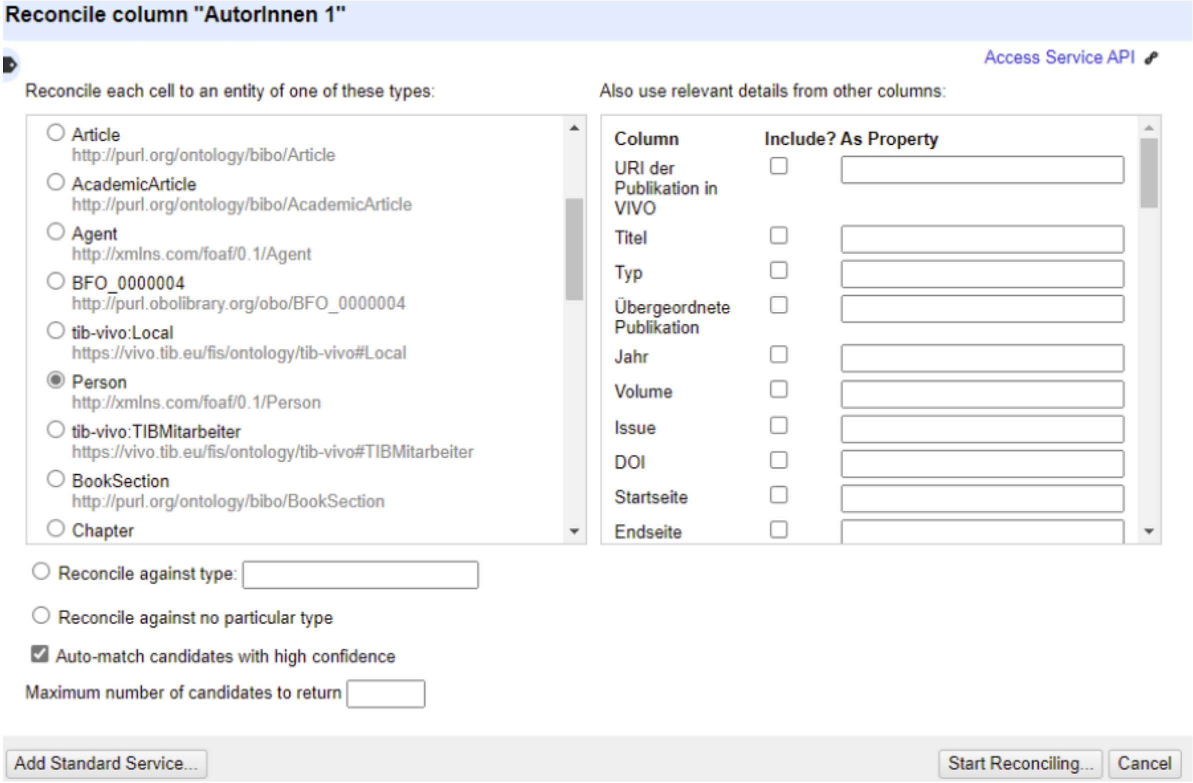
3. Enter the url of the reconciliation service and click on Add Service



Add standard service

Enter the service's URL: https://vivo.tib.eu/fis/reconcile

4. The VIVO Reconciliation Service should now show up in OpenRefine.



Reconcile column "AutorInnen 1"

Access Service API

Reconcile each cell to an entity of one of these types:

- ○ Article http://purl.org/ontology/bibo/Article
- ○ AcademicArticle http://purl.org/ontology/bibo/AcademicArticle
- ○ Agent http://xmlns.com/foaf/0.1/Agent
- ○ BFO_0000004 http://purl.obolibrary.org/obo/BFO_0000004
- ○ tib-vivo:Local https://vivo.tib.eu/fis/ontology/tib-vivo#Local
- ● Person http://xmlns.com/foaf/0.1/Person
- ○ tib-vivo:TIBMitarbeiter https://vivo.tib.eu/fis/ontology/tib-vivo#TIBMitarbeiter
- ○ BookSection http://purl.org/ontology/bibo/BookSection
- ○ Chapter

Also use relevant details from other columns:

Column	Include?	As Property
URI der Publikation in VIVO	☐	
Titel	☐	
Typ	☐	
Übergeordnete Publikation	☐	
Jahr	☐	
Volume	☐	
Issue	☐	
DOI	☐	
Startseite	☐	
Endseite	☐	

○ Reconcile against type:

○ Reconcile against no particular type

☑ Auto-match candidates with high confidence

Maximum number of candidates to return

In the menu you can choose the type in VIVO, against which you want to reconcile your data. Click on the button “Start reconciling” to process the matching.

For further information please consult the OpenRefine documentation: https://docs.openrefine.org/manual/reconciling

10.2.5.4 Configuration of the VIVO Reconciliation API

A list of types that are delivered via the API can be configured in the runtime properties. For instance, the property Vitro.reconcile.defaultTypeList can be configured as follows:

```
http://vivoweb.org/ontology/core#Role, core:Role; http://vivoweb.org/ontology/
core#AcademicDegree, core:Academic Degree; http://purl.org/NET/c4dm/event.owl#Event,
event:Event; http://vivoweb.org/ontology/core#Location, core:Location; http://
xmlns.com/foaf/0.1/Organization, foaf:Organization; http://xmlns.com/foaf/0.1/Person,
foaf:Person; http://purl.obolibrary.org/obo/IAO_0000030, obo:IAO_0000030
```

A documentation of the technical implementation can be found here: Extending Google Refine for VIVO[150]. Please note that this documentation has not been updated in a long time.

10.2.6 Search indexing service

10.2.6.1 Purpose

Permits external applications to request specific updates to the VIVO search index, by providing a list of URIs whose search records may be out of date.

When the VIVO triple-store is updated in a way that bypasses VIVO's internal data channels, the search index will not reflect the updates.

With this service, you can provide a list of URIs whose contents have changed, and request that only those search records be updated. This is usually faster than rebuilding the entire index.

---

[150] https://wiki.lyrasis.org/display/VIVO/Extending+Google+Refine+for+VIVO

10.2.6.2 Use Cases

10.2.6.2.1 Use with ingest tools

The Harvester and similar tools write directly to the VIVO triple-store, bypassing the usual data channels in VIVO. After ingesting, it has been necessary to rebuild the search index so it will reflect the changes in the data. With this service, you can rebuild only part of the index.

Note: when the Harvester and other tools have been modified to use the SPARQL Update API, VIVO will ensure that the search index and inferences are kept in synchronization with the data.

10.2.6.2.2 Loading the triple-store

Some sites use two VIVO instances: a staging instance and a production instance. All ingests occur on the staging instance. Periodically, the triple-store is copied from staging to production. When this is done, you have 3 options:

- Copy the search index files from staging to production to keep it consistent with the triple-store
- Rebuild the search index in production
- Use the Search Indexing service to update specific records in the search index.

10.2.6.3 Indexing and Reasoning

The concerns that apply to the search index will also apply to the state of the inferred triples in the data model. When bypassing the data channels in VIVO, you bypass the semantic reasoner. To compensate for this, you must either

- Request that the reasoner rebuild all of the inferences, using `Recompute Inferences` from the `Site Administration` page, or
- Ensure that the ingested RDF contains all of the triples that you want VIVO to contain, including those that would be provided by the reasoner

In most cases, the time required to re-inference the model is greater than the time required to rebuild the search index. Unfortunately, the reasoning process is not easy to partition. To date, VIVO has no service that would allow you to update the inferences on a limited set of data.

10.2.6.4 Specification

10.2.6.4.1 URL

`[vivo]/searchService/updateUrisInSearch`

10.2.6.4.1.1 Examples:

```
http://vivo.cornell.edu/searchService/updateUrisInSearch
```

```
http://localhost:8080/vivo/searchService/updateUrisInSearch
```

10.2.6.4.2 HTTP Method

The API supports only HTTP POST requests with a content type of `multipart/form-data`.

If the request does not specify an encoding, UTF-8 is assumed.

10.2.6.4.3 Parameters

name	value
`email`	the email address of a VIVO administrator account
`password`	the password of the VIVO administrator account
`other`	One or more content parts, containing URIs to be indexed, separated by white space and/or commas

The name of the file content is unimportant. The API will examine all parts of the request and add any URIs to the list to be indexed. It is common, however, to put the entire list of URIs into a single content part.

10.2.6.4.4 Response Codes

Code	Reason
`200 OK`	Search indexing request was successful.
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.

Code	Reason
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Update API.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.2.6.5 Examples

This example uses the UNIX `curl` command to request updates to the search records of 3 individuals.

```
curl -v --form 'email=testAdmin@mydomain.edu' --form 'password=Password' --form
'uris=@uriList.txt' 'http://localhost:8080/vivo/searchService/updateUrisInSearch'
```

uriList.txt

```
http://vivo.mydomain.edu/individual/n6724
http://vivo.mydomain.edu/individual/n90987
http://vivo.mydomain.edu/individual/n32
```

10.2.6.6 Securing the API

The Search Indexing service is enabled by default. However, it is recommended that you secure the URL `/searchService` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

10.2.7 SPARQL Query API

10.2.7.1 Purpose

Permits external applications to obtain data from the VIVO data model.

The results of the queries are not filtered, so access to the service should remain restricted if the VIVO instance contains any data which should remain private. Queries can be performed against the entire data model, or against specific graphs. .

By default, the SPARQL Query API is disabled in VIVO, for security reasons. See Enabling the API[151]

10.2.7.2 Use Cases

10.2.7.2.1 Reusing data from VIVO

Data in VIVO is available to other applications via Linked Open Data - requests and responses (see page 346). But some applications may work better with the sort of data sets that can be obtained from SPARQL queries.

10.2.7.2.2 Writing a VIVO "face" application

Various VIVO sites have written applications, in Drupal or other such frameworks, that display data from VIVO, and allow the user to edit their data. This API, used in conjunction with SPARQL Update API (see page 372), allows such an application to freely read or modify VIVO data.

10.2.7.3 Specification

10.2.7.3.1 URL

`[vivo]/api/sparqlQuery`

Examples:

```
http://vivo.cornell.edu/api/sparqlQuery
```

---

[151] https://wiki.duraspace.org/display/VIVODOC111x/SPARQL+Query+API#SPARQLQueryAPI-EnablingtheSPARQLQueryAPI

```
http://localhost:8080/vivo/api/sparqlQuery
```

10.2.7.3.2 HTTP Method

The API supports HTTP GET or POST calls.

10.2.7.3.3 Parameters

name	value
`email`	the email address of a VIVO administrator account
`password`	the password of the VIVO administrator account
`query`	A SPARQL query

The syntax of the SPARQL query is described on the World Wide Web Consortium site at http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

10.2.7.3.4 Response Codes

Code	Reason
`200 OK`	SPARQL query was successful.
`400 Bad Request`	HTTP request did not include a `query` parameter.
	The SPARQL query was syntactically incorrect.
	The type of the SPARQL query was not `SELECT`, `ASK`, `CONSTRUCT`, or `DESCRIBE`
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.

Code	Reason
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Query API.
`406 Not Acceptable`	The Accept header does not include any available content types.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.2.7.3.5 Available content types

The request may include an `Accept` header, to specify the preferred content type of the response. If no `Accept` header is provided, the preferred content type is assumed to be `text/plain`.

10.2.7.3.5.1 For `SELECT` or `ASK` queries

`SELECT` queries return rows of results, and each row may include an arbitrary number of values, depending on the query.

`ASK` queries return a single result, which is either `true` or `false`.

MIME type in the Accept header	Response format	Format description
`text/plain`	text	
`text/csv`	CSV	http://www.w3.org/TR/2013/REC-sparql11-results-csv-tsv-20130321
`text/tab-separated-values`	TSV	
`application/sparql-results+xml`	XML	http://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321

MIME type in the Accept header	Response format	Format description
`application/sparql-results+json`	JSON	http://www.w3.org/TR/2013/REC-sparql11-results-json-20130321

10.2.7.3.5.2 For `CONSTRUCT` or DESCRIBE queries

`CONSTRUCT` and `DESCRIBE` queries return RDF.

MIME type in the Accept header	Response format	Format description
`text/plain`	N-Triples	http://www.w3.org/2001/sw/RDFCore/ntriples/
`application/rdf+xml`	RDF/XML	http://www.w3.org/TR/rdf-syntax-grammar/
`text/n3`	N3	http://www.w3.org/TeamSubmission/n3/
`text/turtle`	Turtle	http://www.w3.org/TeamSubmission/turtle/
`application/json`	JSON-LD	http://www.w3.org/TR/json-ld/

10.2.7.3.6 Limitation

Queries can be performed against specific graphs. However, the graphs that hold application data are not accessible to the API. "Application data" means data that controls the functioning of the VIVO application, such as user accounts, page definitions, or display parameters.

10.2.7.4 Examples

These examples use the UNIX `curl` command to issue queries to the API.

10.2.7.4.1 SELECT to JSON example

This example reads 5 arbitrary triples from the data model, returning the result as JSON.

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d 'query=SELECT ?
s ?p ?o WHERE {?s ?p ?o} LIMIT 5' -H 'Accept: application/sparql-results+json'
'http://localhost:8080/vivo/api/sparqlQuery'
```

The response looks like this:

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "bnode" , "value": "b0" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-
ns#rest" } ,
        "o": { "type": "bnode" , "value": "b1" }
      } ,
      {
        "s": { "type": "bnode" , "value": "b0" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-
ns#first" } ,
        "o": { "type": "uri" , "value": "http://purl.obolibrary.org/obo/ERO_0000006"
}
      } ,
      {
        "s": { "type": "bnode" , "value": "b2" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-
ns#rest" } ,
        "o": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil" }
      } ,
      {
        "s": { "type": "bnode" , "value": "b2" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-
ns#first" } ,
        "o": { "type": "bnode" , "value": "b3" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://vivoweb.org/ontology/
core#FacultyMember" } ,
        "p": { "type": "uri" , "value": "http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#hiddenFromDisplayBelowRoleLevelAnnot" } ,
        "o": { "type": "uri" , "value": "http://vitro.mannlib.cornell.edu/ns/vitro/
role#public" }
      }
    ]
  }
}
```

10.2.7.4.2 DESCRIBE to N3 example

This example reads all of the properties for a particular individual in the model, returning the result as N3.

```
curl -i -d 'email=vivo_root@mydomain.edu' -d 'password=Password' -d 'query=DESCRIBE
<http://dbpedia.org/resource/Connecticut>' -H 'Accept: text/n3' 'http://
localhost:8080/vivo/api/sparqlQuery'
```

The response looks like this:

```
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://dbpedia.org/resource/Connecticut>
      a       <http://vivoweb.org/ontology/core#StateOrProvince> ,
              <http://purl.obolibrary.org/obo/BFO_0000006> ,
              <http://vivoweb.org/ontology/core#Location> ,
              owl:Thing ,
              <http://vivoweb.org/ontology/core#GeopoliticalEntity> ,
              <http://purl.obolibrary.org/obo/BFO_0000002> ,
              <http://vivoweb.org/ontology/core#GeographicRegion> ,
              <http://purl.obolibrary.org/obo/BFO_0000001> ,
              <http://purl.obolibrary.org/obo/BFO_0000141> ,
              <http://vivoweb.org/ontology/core#GeographicLocation> ,
              <http://purl.obolibrary.org/obo/BFO_0000004> ;
      <http://www.w3.org/2000/01/rdf-schema#label>
              "Connecticut"@en ;
      <http://purl.obolibrary.org/obo/BFO_0000050>
              <http://aims.fao.org/aos/geopolitical.owl#United_States_of_America> ;
      vitro:mostSpecificType
              <http://vivoweb.org/ontology/core#StateOrProvince> .
```

10.2.7.5 Enabling the SPARQL Query API

Before enabling the SPARQL Query API, you should secure the URL `api/sparqlQuery` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

By default, the SPARQL Query API is disabled in VIVO for all users except the root user. To enable it for non-root users, you must edit the RDF file `[vivo]/home/rdf/auth/everytime/permission_config.n3` to authorize your site administrators to use the API. Find the permissions for `auth:ADMIN` and include the following permission:

permission_config.n3

```
auth:hasPermission simplePermission:UseSparqlQueryApi;
```

After editing this file you need to restart tomcat.

10.2.8 SPARQL Update API

10.2.8.1 Purpose

Permits external applications to add or remove specific triples from the VIVO data model. These changes use the standard data channels in VIVO, so the search index will be updated as appropriate, and the reasoner will add or remove inferences as needed.

By default, the SPARQL Update API is disabled in VIVO, for security reasons. See Enabling the API (see page 380).

10.2.8.2 Use Cases

10.2.8.2.1 Harvester

Previous implementations of the Harvester and similar tools have written directly to the VIVO triple-store, bypassing the usual data channels in VIVO. After ingesting, it was necessary to rebuild the search index, and to run the reasoner to add or remove inferences. Since the search index and the reasoner were not aware of the exact changes, the entire data model was re-indexed and re-inferenced.

When the Harvester and other tools have been modified to use the SPARQL Update API, VIVO will ensure that the search index and inferences are kept in synchronization with the data.

10.2.8.2.2 Other ingest tools

This API permits ingest tools such as Karma to programmatically insert data into VIVO without requiring knowledge of VIVOs internal data structures.

10.2.8.2.3 VIVO "face" applications

Linked Open Data requests have permitted people to write Drupal applications (for example) that display data from VIVO. This API will permit such applications to accept user edits, and apply them back to VIVO.

10.2.8.3 Specification

10.2.8.3.1 URL

`[vivo]/api/sparqlUpdate`

Examples:

```
http://vivo.cornell.edu/api/sparqlUpdate
```

```
http://localhost:8080/vivo/api/sparqlUpdate
```

10.2.8.3.2 HTTP Method

The API supports only HTTP POST calls. GET, HEAD, and other methods are not supported, and will return a response code of `405 Method Not Allowed`.

10.2.8.3.3 Parameters

name	value	notes
`email`	the email address of a VIVO administrator account	
`password`	the password of the VIVO administrator account	
`update`	A SPARQL Update request	Optional. If used, request content type must be set to `application/x-www-form-urlencoded` and the update data must be URL-encoded.

The syntax for a SPARQL Update request is described on the World Wide Web Consortium site at http://www.w3.org/TR/2013/REC-sparql11-update-20130321/

10.2.8.3.4 Limitation

The API requires that you specify a GRAPH in your SPARQL update request. Insertions or deletions to the default graph are not supported.

10.2.8.3.5 Sending data via the POST body vs the 'update' parameter

The API supports sending data via an 'update' parameter. Prior to v1.13.0, this was the only method of sending SPARQL update data. Since 1.13.0, VIVO also supports sending data via the POST message body. To use this method, you must set the request content type to `application/sparql-update`. This method parses the data as an input stream and may offer a performance improvement for large amounts of data.

10.2.8.3.6 Response Codes

Code	Reason
`200 OK`	SPARQL Update was successful.
`400 Bad Request`	HTTP request content type was set to `application/x-www-form-urlencoded` and did not include an `update` parameter.
	The SPARQL Update request did not specify a GRAPH.
	The SPARQL Update request was syntactically incorrect.
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Update API.

Code	Reason
`405 Method Not Allowed`	Incorrect HTTP method; only POST is accepted.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.2.8.4 Examples

These examples use the UNIX `curl` command to insert and delete data using the API.

10.2.8.4.1 Insert example

This example inserts a single RDF statement into the data model.

10.2.8.4.1.1 Insert via the POST body

```
curl -i --request POST 'http://localhost:8080/vivo/api/sparqlUpdate?
email=vivo_root@mydomain.edu&password=Password' \
--header 'Content-Type: application/sparql-update' \
--data-raw 'INSERT DATA { GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>
{
 <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
} }'
```

10.2.8.4.1.2 Insert using the 'update' parameter

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@insert.sparql'
'http://localhost:8080/vivo/api/sparqlUpdate'
```

insert.sparql

```
update=INSERT DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
    }
}
```

10.2.8.4.2 Modify example

This example removes the previous statement, and inserts a replacement.

10.2.8.4.2.1 Modify via the POST body

```
curl -i --request POST 'http://localhost:8080/vivo/api/sparqlUpdate?
email=vivo_root@mydomain.edu&password=Password' \
--header 'Content-Type: application/sparql-update' \
--data-raw 'DELETE DATA { GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>
{
 <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
} }

INSERT DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Design Patterns" .
    }
}'
```

10.2.8.4.2.2 Modify via the update parameter

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@modify.sparql'
'http://localhost:8080/vivo/api/sparqlUpdate'
```

modify.sparql

```
update=DELETE DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
    }
}
INSERT DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Design Patterns" .
    }
}
```

10.2.8.4.3 Delete example

This example removes the modified statement.

10.2.8.4.3.1 Delete via the POST body

```
curl -i --request POST 'http://localhost:8080/vivo/api/sparqlUpdate?
email=vivo_root@mydomain.edu&password=Password' \
--header 'Content-Type: application/sparql-update' \
--data-raw 'DELETE DATA { GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>
{
 <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Design Patterns" .
} }'
```

10.2.8.4.3.2 Delete via the update parameter

```
 curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@delete.sparql'
 'http://localhost:8080/vivo/api/sparqlUpdate'
```

delete.sparql

```
update=DELETE DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Design Patterns" .
    }
}
```

10.2.8.4.4 Clear example

If you want to remove a named graph entirely, you can use the SPARQL CLEAR[152] method.

10.2.8.4.4.1 Clear via the POST body

```
curl -i --request POST 'http://localhost:8080/vivo/api/sparqlUpdate?
email=vivo_root@mydomain.edu&password=Password' \
--header 'Content-Type: application/sparql-update' \
--data-raw 'CLEAR GRAPH IRIRef'
```

10.2.8.4.4.2 Clear via the update parameter

CLEAR example

```
curl -i -d 'email=USER' -d 'password=PASSWORD' -d 'update=CLEAR GRAPH IRIRef'
'http://localhost:8080/vivo/api/sparqlUpdate'
```

Replace IRIRef with the URI of the named graph you want to delete, e.g. <http://localhost/data/people>

10.2.8.4.5 Large Files

For large files one can also use the SPARQL LOAD[153] command.

For this, you have to first create the RDF file with the triples that you want to add, and make the file accessible at a URL. In the example below, the RDF file containing the triples is called `data.rdf`, and is available in the root directory of the web server at `myserver.address.xxx`.

Like the previous commands, this one references a data file, in this case called `import.sparql`. That file contains the `LOAD` command which references the actual data.

```
curl -d 'email=USER' -d 'password=PASSWORD' -d '@import.sparql' 'http://
localhost:8080/vivo/api/sparqlUpdate'
```

---

[152] https://www.w3.org/TR/sparql11-update/#clear
[153] http://www.w3.org/TR/sparql11-update/#load

import.sparql

```
update=LOAD <http://myserver.address.xxx/data.rdf> into graph <http://
vitro.mannlib.cornell.edu/default/vitro-kb-2>
```

10.2.8.4.5.1 Increase the default Tomcat `maxPostSize`

By default, Tomcat sets the default maximum of a POST request to 2 megabytes. If you want to increase this to be able to POST larger sets of triples to VIVO, you can use the `maxPostSize` attribute in `server.xml`. The example below would increase the maximum to 10 MB. See the Tomcat documentation[154] for more details.

server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
   URIEncoding="UTF-8"
   redirectPort="8443"
   maxPostSize="10485760"/>
```

10.2.8.4.6 Advanced Use

Since v1.13.0, VIVO supports the use of the advanced parameters `using-graph-uri` and `using-named-graph-uri`. You could use these parameters to query a graph, while simultaneously creating new triples based on the query results. For example:

```
curl --location --request POST 'http://localhost:8080/vivo/api/sparqlUpdate?
email=vivo_root@mydomain.edu&password=Password&using-graph-uri=http://
vitro.mannlib.cornell.edu/default/vitro-kb-2' \
--header 'Content-Type: application/sparql-update' \
--data-raw 'INSERT {
  GRAPH <http://example.com/new-test-graph> {
    ?s <http://example.com/test> "using-graph-uri worked!" .
  }
}
WHERE
  {  ?s  a <http://purl.org/ontology/bibo/Journal> }'
```

---

[154] https://tomcat.apache.org/tomcat-7.0-doc/config/http.html

would query the kb-2 graph for subjects of type bibo:Journal, then insert a new triple into the http://example.com/new-test-graph graph based on the resulting subject URIs. Some discussion on using these parameters can be found on this[155] Stack Overflow page.

10.2.8.4.7 Enabling the API

Before enabling the SPARQL update handler, you should secure the URL `api/sparqlUpdate` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

By default, the SPARQL Update handler is enabled for only the root user in VIVO. To enable it for other user groups, you can either:

- uncomment the line references "UseSparqUpdateAPI" in [vitro]/rdf/auth/everytime/permission_config.n3 or
- create an RDF file in the `[vitro]/rdf/auth/everytime` directory that will authorize your site administrators to use the API. Below is an example of such a file, using N3 syntax.

authorizeSparqlUpdate.n3

```
@prefix auth: <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#> .
@prefix simplePermission:
<java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#> .

# Authorize the ADMIN role to use the SPARQL Update API
auth:ADMIN auth:hasPermission simplePermission:UseSparqlUpdateApi .
```

10.2.9 Triple Pattern Fragments

10.2.9.1 Background

Triple Pattern Fragments is a form of Linked Data Fragments (see References) for querying a triple store to retrieve a set of triples matching a specified pattern. The pattern is always of the form subject predicate object, where any or all of the elements of the pattern may be unspecified, that is, wildcards.

SPARQL and Linked Data Fragments, alternatives to Triple Pattern Fragments, are very powerful, and as a result, can generate queries that take a long time to run, slowing the server, and in some cases, making the server unavailable. This results in sites shutting down their access points for fear of losing availability to long-running queries. Triple Pattern Fragments solves this problem by allowing only one kind of query, the pattern. Pattern matching is indexed and very fast, insuring the servers remain available while handling queries.

---

[155] https://stackoverflow.com/questions/46329168/insert-from-two-named-graphs/46370944#46370944

For example the pattern <uri> * * finds all the triples which have the specified URI as a subject. The pattern * <uri> * finds all the triples with the specified predicate, and * * <uri> finds all the triples with the specified object.

Triple pattern fragments is a very fast, very simple means for querying a triple store. The triple pattern fragments API in VIVO puts little load on the server, providing a simple means for getting data from the triple store. The API has a web interface for manual use, can be used from the command line via curl, and can be used by programs. Each mode of usage is described below.

Open API

Triple Pattern Fragments, as delivered in VIVO, is an open API. This means that anyone, and any software can access the Triple Pattern Fragments endpoint of your VIVO without logging on, that is, without authorization. All the data in your full graph is accessible to the API and to those who use it. VIVO is built for data sharing, and the Triple Pattern Fragments API makes it very easy for your VIVO to share data with others. **Please be sure your VIVO does not contain restricted data that should not be shared with others**

VIVO uses the LinkedDataFragments Server, available on GitHub here: https://github.com/LinkedDataFragments/Server.Java

10.2.9.2 Configuration

There are currently two supported means of exposing a TPF endpoint for the content in VIVO:

1. Enable the embedded TPF server
2. Setup and external TPF server

10.2.9.2.1 Embedded TPF server

Since the TPF endpoint does not currently enforce visibility settings for classes or properties set in VIVO, the service is turned off by default. In order to enable the TPF endpoint, a "tpf.activeFlag" property must be added to the runtime.properties file with a value of "true". For this property to take effect, the Tomcat server must be restarted.

```
tpf.activeFlag = true
```

- Navigate to: http://localhost:8080/vivo/tpf/core

10.2.9.2.2 External TPF server

In order to run the external TPF server, the following steps should be taken:

1. Clone and build the dependency project: https://github.com/rdfhdt/hdt-java
   a.
   ```
   $ git clone https://github.com/rdfhdt/hdt-java -b v2.1
   $ cd hdt-java
   $ mvn install
   ```
   b. Note: This step will not be necessary once the HDT artifact is published to Maven Central
2. Clone and build: https://github.com/LinkedDataFragments/Server.Java
   a.
   ```
   $ git clone https://github.com/LinkedDataFragments/Server.Java.git
   $ cd Server.java
   $ mvn install
   ```
   b. Note: This step will not be necessary once an executable and/or war release has been published
3. Create "config.json" along the lines of the below:

```
{
  "title": "VIVO Linked Data Fragments server",

  "datasourcetypes": {
    "SparqlDatasource"   :
"org.linkeddatafragments.datasource.sparql.SparqlDataSourceType"
  },

  "datasources": {
    "vivo": {
      "title": "VIVO Linked Data TPF",
      "type": "SparqlDatasource",
      "description": "VIVO Semantic TPF server",
      "settings": { "endpoint": "http://localhost:8080/vivo/api/
sparqlQuery",
                    "username": "vivo_root@mydomain.edu",
                    "password": "your-password" }
    }
  },

  "prefixes": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "hydra": "http://www.w3.org/ns/hydra/core#",
    "void": "http://rdfs.org/ns/void#",
    "bibo": "http://purl.org/ontology/bibo/",
    "c4o": "http://purl.org/spar/c4o/",
    "cito": "http://purl.org/spar/cito/",
    "dcterms": "http://purl.org/dc/terms/",
    "event": "http://purl.org/NET/c4dm/event.owl#",
```

```
    "fabio": "http://purl.org/spar/fabio/",
    "foaf": "http://xmlns.com/foaf/0.1/",
    "geo": "http://aims.fao.org/aos/geopolitical.owl#",
    "obo": "http://purl.obolibrary.org/obo/",
    "ocrer": "http://purl.org/net/OCRe/research.owl#",
    "ocresst": "http://purl.org/net/OCRe/statistics.owl#",
    "ocresd": "http://purl.org/net/OCRe/study_design.owl#",
    "ocresp": "http://purl.org/net/OCRe/study_protocol.owl#",
    "ro": "http://purl.obolibrary.org/obo/ro.owl#",
    "skos": "http://www.w3.org/2004/02/skos/core#",
    "swo": "http://www.ebi.ac.uk/efo/swo/",
    "vcard": "http://www.w3.org/2006/vcard/ns#",
    "vitro-public": "http://vitro.mannlib.cornell.edu/ns/vitro/
public#",
    "vivo": "http://vivoweb.org/ontology/core#",
    "scires": "http://vivoweb.org/ontology/scientific-research#",
    "vann": "http://purl.org/vocab/vann/"
  }
}
```

4. Deploy the externalized TPF server:
    a. As a standalone application:

    ```
    $ java -jar target/ldf-server.jar config.json --port <port-number | 8080>
    ```

    b. Or, as a standard web-application, deployed in Tomcat or Jetty
        i. Place 'target/ldf-server.war' in your servlet container of choice
5. Navigate to: http://localhost:[port-number]/vivo[156]

10.2.9.3 Manual Query

Manual query can be used to view triples, and to run SPARQL queries resolved as Triple Pattern Fragments. These methods are view data. To save data, use curl, or Programmatic Access.

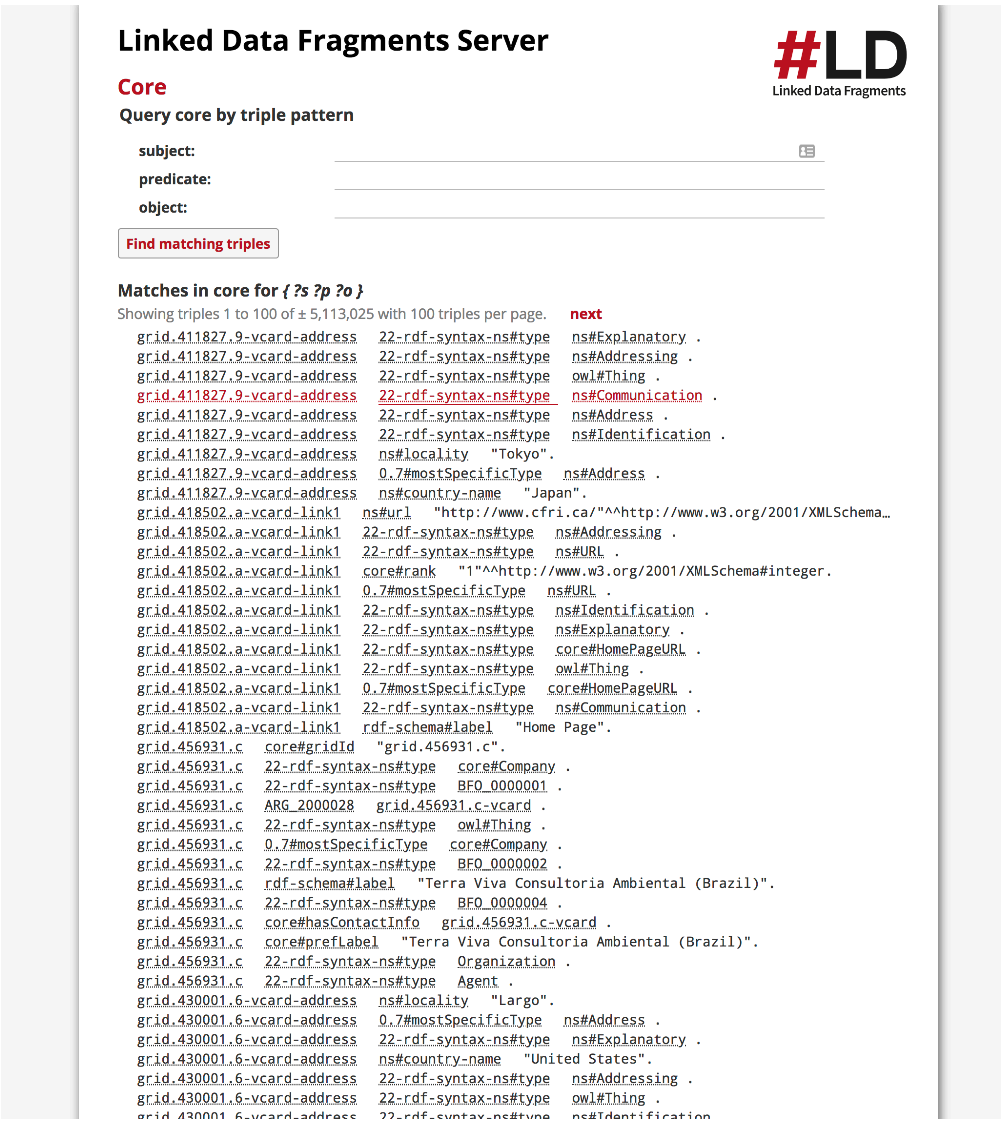
10.2.9.3.1 View Triples using Triple Pattern Fragments

To use TPF manually, visit the TPF endpoint of a VIVO, http://yourvivo/tpf/core,[157] where "yourvivo" is the web address of the VIVO of interest. In the example below we use http://openvivo.org/tpf/core. You will see:

---

[156] http://localhost:7070/vivo

[157] http://vivo/tpf/core,



Linked Data Fragments Server

Core

Query core by triple pattern

subject:
predicate:
object:

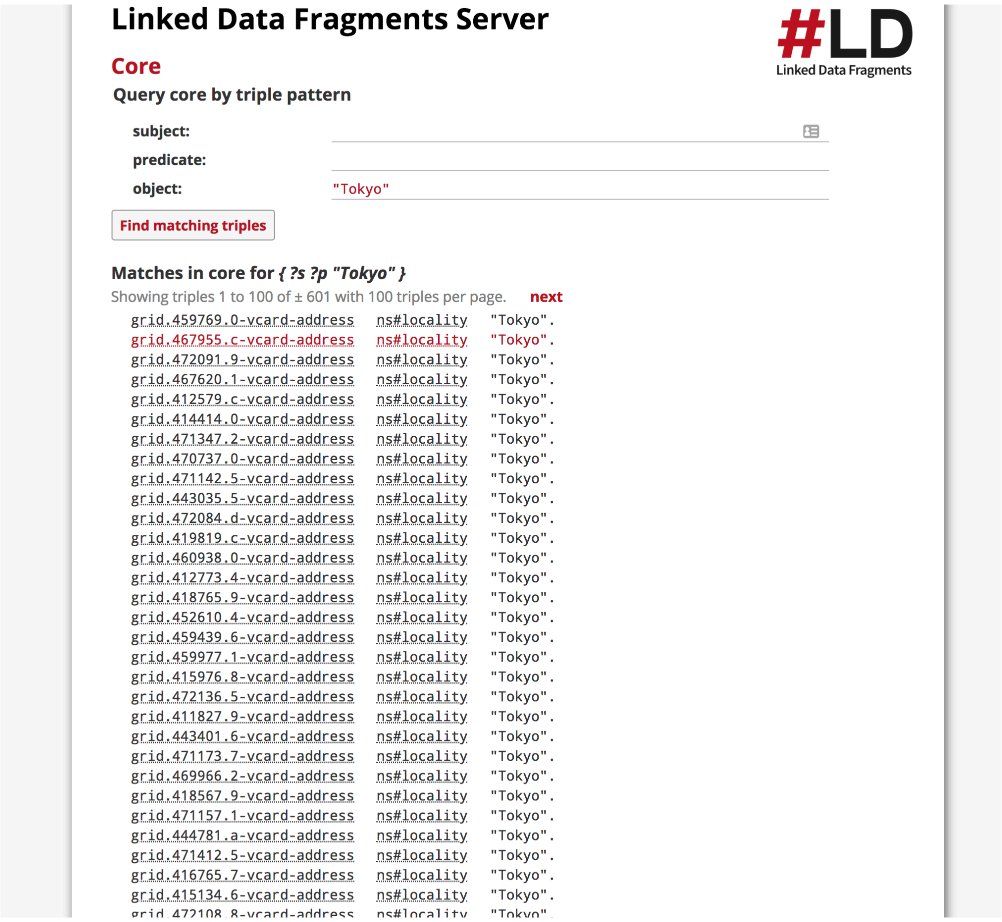
Matches in core for *{ ?s ?p ?o }*

Showing triples 1 to 100 of ± 5,113,025 with 100 triples per page. **next**

```
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  ns#Explanatory .
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  ns#Addressing .
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  owl#Thing .
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  ns#Communication .
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  ns#Address .
grid.411827.9-vcard-address  22-rdf-syntax-ns#type  ns#Identification .
grid.411827.9-vcard-address  ns#locality  "Tokyo".
grid.411827.9-vcard-address  0.7#mostSpecificType  ns#Address .
grid.411827.9-vcard-address  ns#country-name  "Japan".
grid.418502.a-vcard-link1  ns#url  "http://www.cfri.ca/"^^http://www.w3.org/2001/XMLSchema…
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  ns#Addressing .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  ns#URL .
grid.418502.a-vcard-link1  core#rank  "1"^^http://www.w3.org/2001/XMLSchema#integer.
grid.418502.a-vcard-link1  0.7#mostSpecificType  ns#URL .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  ns#Identification .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  ns#Explanatory .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  core#HomePageURL .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  owl#Thing .
grid.418502.a-vcard-link1  0.7#mostSpecificType  core#HomePageURL .
grid.418502.a-vcard-link1  22-rdf-syntax-ns#type  ns#Communication .
grid.418502.a-vcard-link1  rdf-schema#label  "Home Page".
grid.456931.c  core#gridId  "grid.456931.c".
grid.456931.c  22-rdf-syntax-ns#type  core#Company .
grid.456931.c  22-rdf-syntax-ns#type  BFO_0000001 .
grid.456931.c  ARG_2000028  grid.456931.c-vcard .
grid.456931.c  22-rdf-syntax-ns#type  owl#Thing .
grid.456931.c  0.7#mostSpecificType  core#Company .
grid.456931.c  22-rdf-syntax-ns#type  BFO_0000002 .
grid.456931.c  rdf-schema#label  "Terra Viva Consultoria Ambiental (Brazil)".
grid.456931.c  22-rdf-syntax-ns#type  BFO_0000004 .
grid.456931.c  core#hasContactInfo  grid.456931.c-vcard .
grid.456931.c  core#prefLabel  "Terra Viva Consultoria Ambiental (Brazil)".
grid.456931.c  22-rdf-syntax-ns#type  Organization .
grid.456931.c  22-rdf-syntax-ns#type  Agent .
grid.430001.6-vcard-address  ns#locality  "Largo".
grid.430001.6-vcard-address  0.7#mostSpecificType  ns#Address .
grid.430001.6-vcard-address  22-rdf-syntax-ns#type  ns#Explanatory .
grid.430001.6-vcard-address  ns#country-name  "United States".
grid.430001.6-vcard-address  22-rdf-syntax-ns#type  ns#Addressing .
grid.430001.6-vcard-address  22-rdf-syntax-ns#type  owl#Thing .
grid.430001.6-vcard-address  22-rdf-syntax-ns#type  ns#Identification
```

Notice that results are returned for the triple pattern fragment * * *. More than 5 million triples were returned, with the first hundred being displayed on the web page. Pressing "next" at the top of the list of triples will display the next hundred triples. Each of the rows in the display is a triple in the full graph. The TPF web page uses a display with simplifies the presentation of URI. Many of the URI are shortened, not with the use of prefixes, but merely by intelligently truncating the URI.

Each of the elements in each of the triples is a link. Clicking a link will issue a TPF query with the selected element as the specified value in a triple. For example, clicking on "Tokyo" in the example above, generates a TPF request of the form * * "Tokyo" – that is, find all the triples that have the text string "Tokyo" as an object. The result is shown below:



Linked Data Fragments Server

Core

Query core by triple pattern

subject:

predicate:

object: "Tokyo"

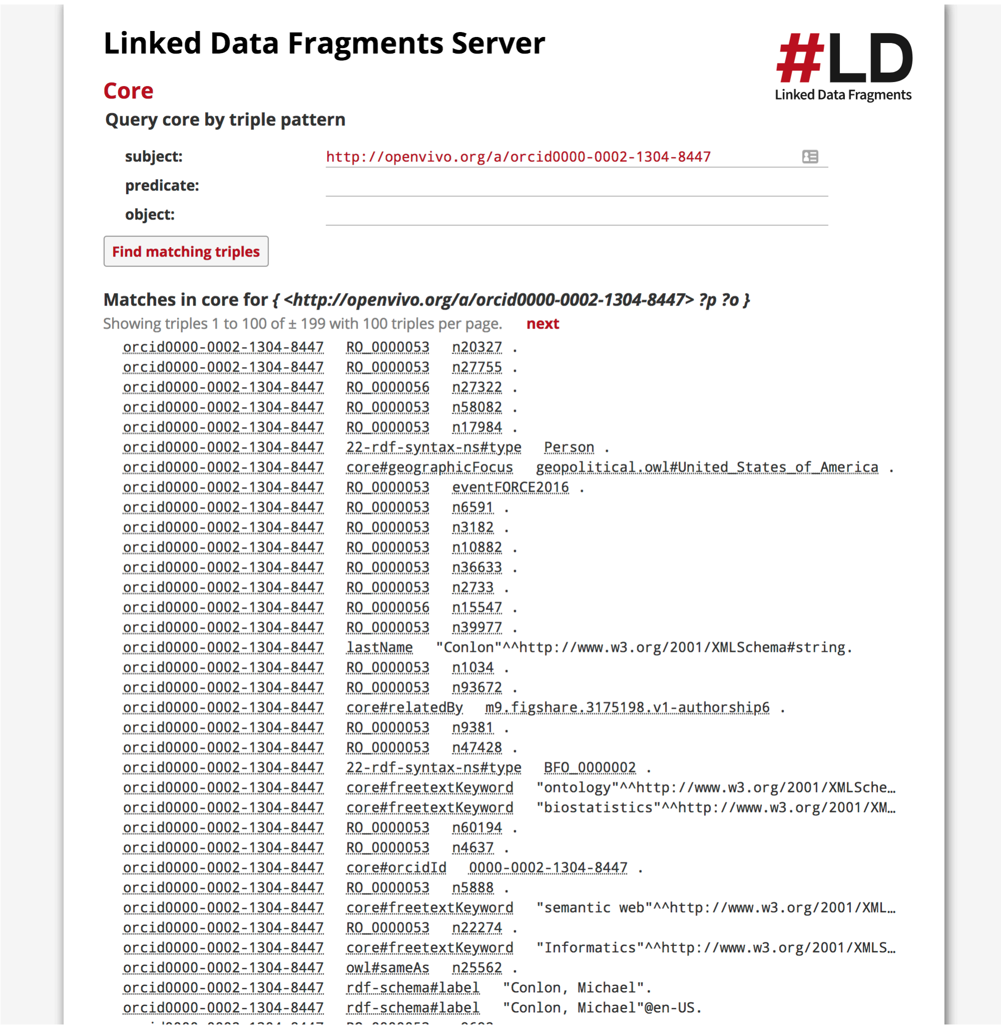
Matches in core for *{ ?s ?p "Tokyo" }*

Showing triples 1 to 100 of ± 601 with 100 triples per page.

```
grid.459769.0-vcard-address   ns#locality   "Tokyo".
grid.467955.c-vcard-address   ns#locality   "Tokyo".
grid.472091.9-vcard-address   ns#locality   "Tokyo".
grid.467620.1-vcard-address   ns#locality   "Tokyo".
grid.412579.c-vcard-address   ns#locality   "Tokyo".
grid.414414.0-vcard-address   ns#locality   "Tokyo".
grid.471347.2-vcard-address   ns#locality   "Tokyo".
grid.470737.0-vcard-address   ns#locality   "Tokyo".
grid.471142.5-vcard-address   ns#locality   "Tokyo".
grid.443035.5-vcard-address   ns#locality   "Tokyo".
grid.472084.d-vcard-address   ns#locality   "Tokyo".
grid.419819.c-vcard-address   ns#locality   "Tokyo".
grid.460938.0-vcard-address   ns#locality   "Tokyo".
grid.412773.4-vcard-address   ns#locality   "Tokyo".
grid.418765.9-vcard-address   ns#locality   "Tokyo".
grid.452610.4-vcard-address   ns#locality   "Tokyo".
grid.459439.6-vcard-address   ns#locality   "Tokyo".
grid.459977.1-vcard-address   ns#locality   "Tokyo".
grid.415976.8-vcard-address   ns#locality   "Tokyo".
grid.472136.5-vcard-address   ns#locality   "Tokyo".
grid.411827.9-vcard-address   ns#locality   "Tokyo".
grid.443401.6-vcard-address   ns#locality   "Tokyo".
grid.471173.7-vcard-address   ns#locality   "Tokyo".
grid.469966.2-vcard-address   ns#locality   "Tokyo".
grid.418567.9-vcard-address   ns#locality   "Tokyo".
grid.471157.1-vcard-address   ns#locality   "Tokyo".
grid.444781.a-vcard-address   ns#locality   "Tokyo".
grid.471412.5-vcard-address   ns#locality   "Tokyo".
grid.416765.7-vcard-address   ns#locality   "Tokyo".
grid.415134.6-vcard-address   ns#locality   "Tokyo".
grid.472108.8-vcard-address   ns#locality   "Tokyo".
```

We see that OpenVIVO has 601 triples with "Tokyo" as an object. The displayed triples show Tokyo as a locality in an address. Putting double quotes around literal values is required. TPF supports the use of language tags to select literal values with specific language tags.

To specify a URI in a pattern, give the full URI (no prefix, no truncation) with no brackets. For example, to find all the triples with a subject of http://openvivo.org/a/orcid0000-0002-1304-8447, put the URI in the subject field and leave predicate field and the object field empty. See below:



Linked Data Fragments Server

Core

Query core by triple pattern

subject: http://openvivo.org/a/orcid0000-0002-1304-8447

predicate:

object:

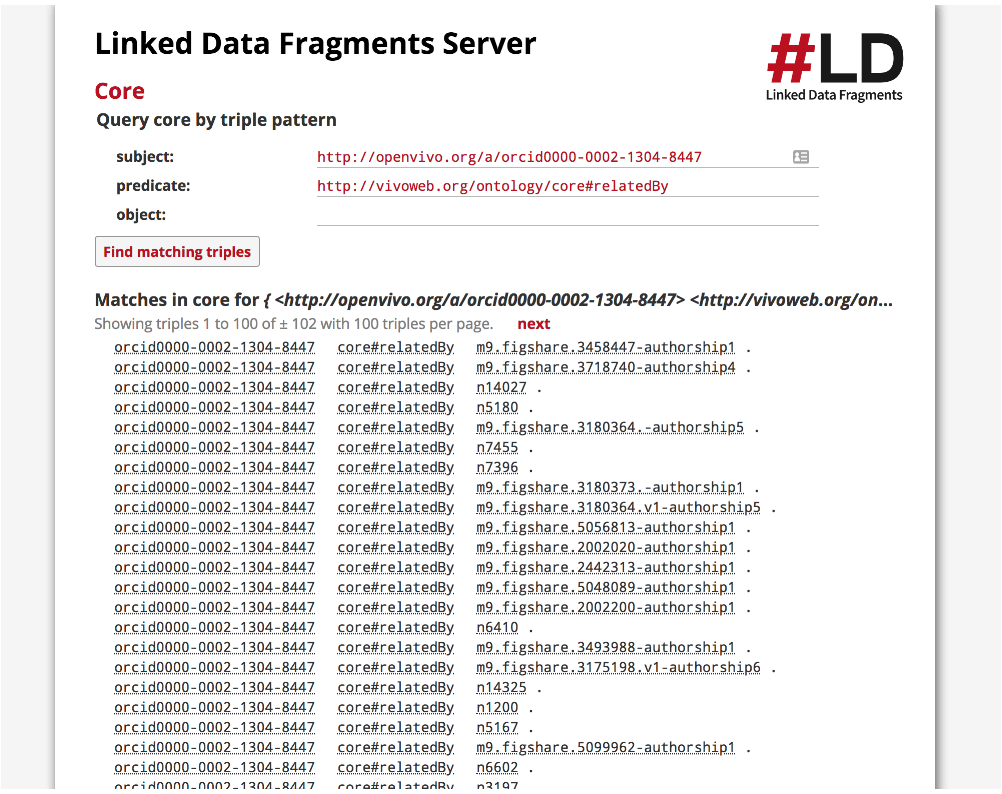
Matches in core for *{ <http://openvivo.org/a/orcid0000-0002-1304-8447> ?p ?o }*

Showing triples 1 to 100 of ± 199 with 100 triples per page. **next**

```
orcid0000-0002-1304-8447   RO_0000053   n20327 .
orcid0000-0002-1304-8447   RO_0000053   n27755 .
orcid0000-0002-1304-8447   RO_0000056   n27322 .
orcid0000-0002-1304-8447   RO_0000053   n58082 .
orcid0000-0002-1304-8447   RO_0000053   n17984 .
orcid0000-0002-1304-8447   22-rdf-syntax-ns#type   Person .
orcid0000-0002-1304-8447   core#geographicFocus   geopolitical.owl#United_States_of_America .
orcid0000-0002-1304-8447   RO_0000053   eventFORCE2016 .
orcid0000-0002-1304-8447   RO_0000053   n6591 .
orcid0000-0002-1304-8447   RO_0000053   n3182 .
orcid0000-0002-1304-8447   RO_0000053   n10882 .
orcid0000-0002-1304-8447   RO_0000053   n36633 .
orcid0000-0002-1304-8447   RO_0000053   n2733 .
orcid0000-0002-1304-8447   RO_0000056   n15547 .
orcid0000-0002-1304-8447   RO_0000053   n39977 .
orcid0000-0002-1304-8447   lastName   "Conlon"^^http://www.w3.org/2001/XMLSchema#string.
orcid0000-0002-1304-8447   RO_0000053   n1034 .
orcid0000-0002-1304-8447   RO_0000053   n93672 .
orcid0000-0002-1304-8447   core#relatedBy   m9.figshare.3175198.v1-authorship6 .
orcid0000-0002-1304-8447   RO_0000053   n9381 .
orcid0000-0002-1304-8447   RO_0000053   n47428 .
orcid0000-0002-1304-8447   22-rdf-syntax-ns#type   BFO_0000002 .
orcid0000-0002-1304-8447   core#freetextKeyword   "ontology"^^http://www.w3.org/2001/XMLSche…
orcid0000-0002-1304-8447   core#freetextKeyword   "biostatistics"^^http://www.w3.org/2001/XM…
orcid0000-0002-1304-8447   RO_0000053   n60194 .
orcid0000-0002-1304-8447   RO_0000053   n4637 .
orcid0000-0002-1304-8447   core#orcidId   0000-0002-1304-8447 .
orcid0000-0002-1304-8447   RO_0000053   n5888 .
orcid0000-0002-1304-8447   core#freetextKeyword   "semantic web"^^http://www.w3.org/2001/XML…
orcid0000-0002-1304-8447   RO_0000053   n22274 .
orcid0000-0002-1304-8447   core#freetextKeyword   "Informatics"^^http://www.w3.org/2001/XMLS…
orcid0000-0002-1304-8447   owl#sameAs   n25562 .
orcid0000-0002-1304-8447   rdf-schema#label   "Conlon, Michael".
orcid0000-0002-1304-8447   rdf-schema#label   "Conlon, Michael"@en-US.
```

199 triples are returned. Each has the specified subject. Data managers might note:

1. There are many RO_0000053 predicates. These are "bearer_of" role assertions. See Ontology Reference (see page 446) for diagrams showing how VIVO uses roles and represents information. TPF can be a very good tools for exploring the data and learning about information representation.
2. The sixth triple is an assertion that the subject in question is a person. At the bottom of the screen shot we see that the person has two labels. "Michael Conlon" and "Michael Conlon"@en-US. Is this something that is expected, or something that should be changed? TPF makes a good tool for discussing VIVO data practices with others.
3. We see that the person is relatedBy an authorship. How many relatedBy assertions does this person have? We could issue a TPF query for the subject and for related by as a predicate. The results are shown below:



Linked Data Fragments Server

Core

Query core by triple pattern

subject: http://openvivo.org/a/orcid0000-0002-1304-8447
predicate: http://vivoweb.org/ontology/core#relatedBy
object:

Matches in core for *{ <http://openvivo.org/a/orcid0000-0002-1304-8447> <http://vivoweb.org/on...*

Showing triples 1 to 100 of ± 102 with 100 triples per page.

```
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3458447-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3718740-authorship4 .
orcid0000-0002-1304-8447  core#relatedBy  n14027 .
orcid0000-0002-1304-8447  core#relatedBy  n5180 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3180364.-authorship5 .
orcid0000-0002-1304-8447  core#relatedBy  n7455 .
orcid0000-0002-1304-8447  core#relatedBy  n7396 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3180373.-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3180364.v1-authorship5 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.5056813-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.2002020-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.2442313-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.5048089-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.2002200-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  n6410 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3493988-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.3175198.v1-authorship6 .
orcid0000-0002-1304-8447  core#relatedBy  n14325 .
orcid0000-0002-1304-8447  core#relatedBy  n1200 .
orcid0000-0002-1304-8447  core#relatedBy  n5167 .
orcid0000-0002-1304-8447  core#relatedBy  m9.figshare.5099962-authorship1 .
orcid0000-0002-1304-8447  core#relatedBy  n6602 .
orcid0000-0002-1304-8447  core#relatedBy  n3197
```

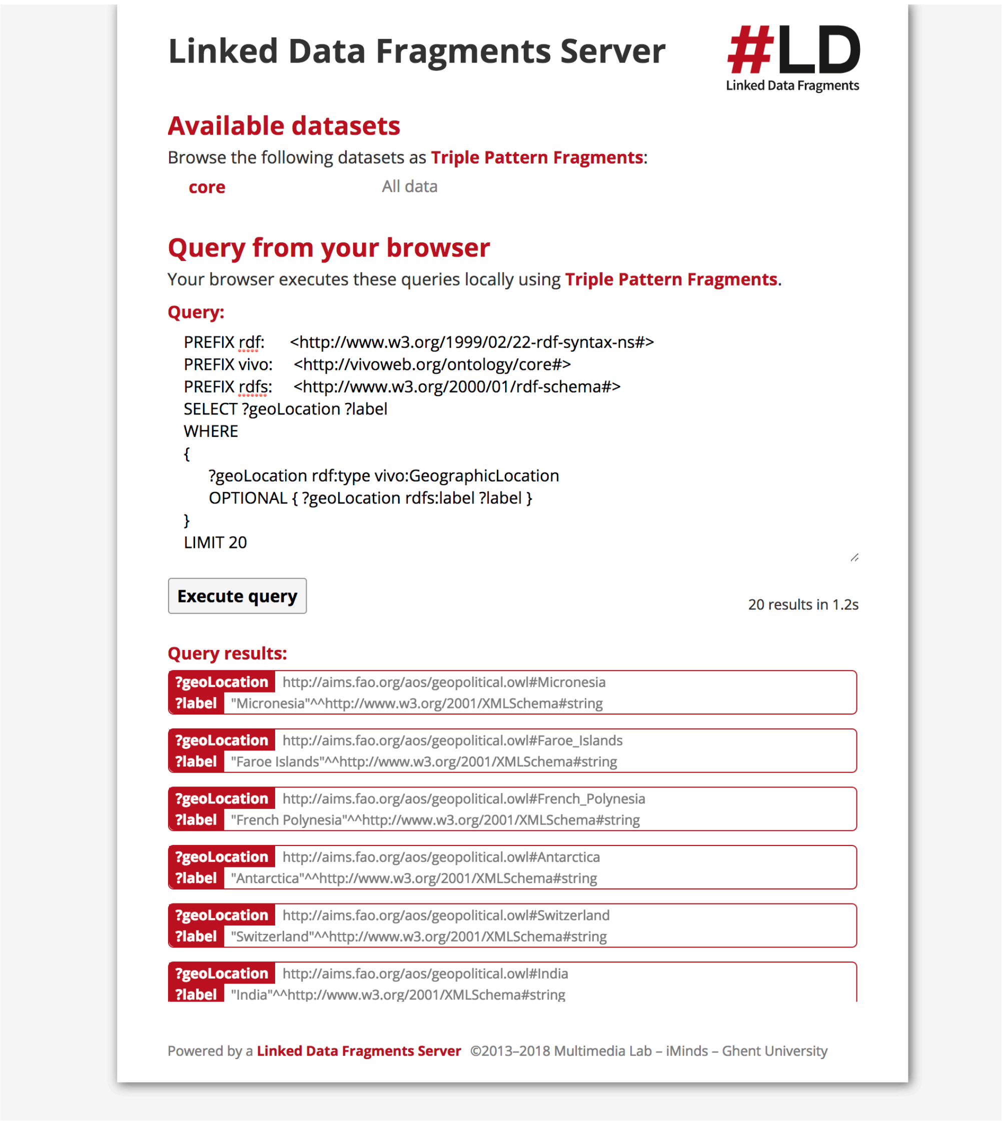
We see 102 triples are returned. Each indicates that the person is relatedBy to something else. We see some of the objects appear to be figshare related, others appear to be authorships, while still others non informative. Additional exploration might help us understand how the relatedBy assertions are used.

10.2.9.3.2 SPARQL Queries Resolved as Triple Pattern Fragments

The Linked Data Fragments server can also resolve full SPARQL queries. The queries are decomposed into a series of TPF requests behind the scenes in the browser. The VIVO server sees only TPF requests. Each TPF request is handled quickly as previously described. To issue a SPARQL query using Triple Pattern Fragments, go to http://yourvivo/tpf where "yourvivo" is the web address of your VIVO. You will see a screen such as that below. Type in your query. You will need to provide the prefixes used in your query, as shown below. Press Execute, and the query is resolved a series of TPF queries. Results are presented dynamically.

To try this yourself, you can use the Linked Data Fragments Server of OpenVIVO, available here: http://openvivo.org/tpf

Note in the example that prefixes are supplied as part of the query. The Triple Pattern Fragments server has no knowledge of VIVO prefixes. These must be supplied with the query.



Linked Data Fragments Server

Available datasets

Browse the following datasets as **Triple Pattern Fragments**:

core All data

Query from your browser

Your browser executes these queries locally using **Triple Pattern Fragments**.

Query:

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vivo:  <http://vivoweb.org/ontology/core#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?geoLocation ?label
WHERE
{
    ?geoLocation rdf:type vivo:GeographicLocation
    OPTIONAL { ?geoLocation rdfs:label ?label }
}
LIMIT 20
```

20 results in 1.2s

Query results:

```
?geoLocation http://aims.fao.org/aos/geopolitical.owl#Micronesia
?label "Micronesia"^^http://www.w3.org/2001/XMLSchema#string

?geoLocation http://aims.fao.org/aos/geopolitical.owl#Faroe_Islands
?label "Faroe Islands"^^http://www.w3.org/2001/XMLSchema#string

?geoLocation http://aims.fao.org/aos/geopolitical.owl#French_Polynesia
?label "French Polynesia"^^http://www.w3.org/2001/XMLSchema#string

?geoLocation http://aims.fao.org/aos/geopolitical.owl#Antarctica
?label "Antarctica"^^http://www.w3.org/2001/XMLSchema#string

?geoLocation http://aims.fao.org/aos/geopolitical.owl#Switzerland
?label "Switzerland"^^http://www.w3.org/2001/XMLSchema#string

?geoLocation http://aims.fao.org/aos/geopolitical.owl#India
?label "India"^^http://www.w3.org/2001/XMLSchema#string
```

Powered by a **Linked Data Fragments Server** ©2013–2018 Multimedia Lab – iMinds – Ghent University

10.2.9.4 Curl

curl can be used to issue triple pattern fragment queries and return RDF/XML. For example:

```
curl http://openvivo.org/tpf/core
```

returns 169 lines of RDF/XML, output truncated below

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    ...
    xmlns:vivo="http://vivoweb.org/ontology/core#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:vitro-public="http://vitro.mannlib.cornell.edu/ns/vitro/public#">
```

```
  <void:Dataset rdf:about="http://openvivo.org/tpf/core#dataset">
    <hydra:search rdf:parseType="Resource">
      <hydra:template>http://openvivo.org/tpf/core{?subject,predicate,object}</hydra:template>
      <hydra:mapping rdf:parseType="Resource">
        <hydra:variable>subject</hydra:variable>
        <hydra:property rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#subject"/>
      </hydra:mapping>
      <hydra:mapping rdf:parseType="Resource">
        <hydra:variable>predicate</hydra:variable>
        <hydra:property rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate"/>
      </hydra:mapping>
      <hydra:mapping rdf:parseType="Resource">
        <hydra:variable>object</hydra:variable>
        <hydra:property rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#object"/>
      </hydra:mapping>
    </hydra:search>
    <rdf:type rdf:resource="http://www.w3.org/ns/hydra/core#Collection"/>
    <void:subset>
      <hydra:Collection rdf:about="http://openvivo.org/tpf/core">
        <hydra:firstPage rdf:resource="http://openvivo.org/tpf/core?page=1"/>
        <hydra:nextPage rdf:resource="http://openvivo.org/tpf/core?page=2"/>
        <rdf:type rdf:resource="http://www.w3.org/ns/hydra/core#PagedCollection"/>
        <void:triples rdf:datatype="http://www.w3.org/2001/XMLSchema#integer"
        >5113025</void:triples>
        <hydra:totalItems rdf:datatype="http://www.w3.org/2001/XMLSchema#integer"
        >5113025</hydra:totalItems>
        <hydra:itemsPerPage rdf:datatype="http://www.w3.org/2001/XMLSchema#integer"
        >100</hydra:itemsPerPage>
      </hydra:Collection>
    </void:subset>
    <hydra:itemsPerPage rdf:datatype="http://www.w3.org/2001/XMLSchema#long"
    >100</hydra:itemsPerPage>
  </void:Dataset>
  <vivo:Company rdf:about="http://openvivo.org/a/grid.456931.c">
    <vivo:gridId>grid.456931.c</vivo:gridId>
    <rdf:type rdf:resource="http://purl.obolibrary.org/obo/BFO_0000001"/>
    <obo:ARG_2000028 rdf:resource="http://openvivo.org/a/grid.456931.c-vcard"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <j.0:mostSpecificType rdf:resource="http://vivoweb.org/ontology/core#Company"/>
    <rdf:type rdf:resource="http://purl.obolibrary.org/obo/BFO_0000002"/>
    <rdfs:label>Terra Viva Consultoria Ambiental (Brazil)</rdfs:label>
    <rdf:type rdf:resource="http://purl.obolibrary.org/obo/BFO_0000004"/>
    <vivo:hasContactInfo rdf:resource="http://openvivo.org/a/grid.456931.c-vcard"/>
    <skos:prefLabel>Terra Viva Consultoria Ambiental (Brazil)</skos:prefLabel>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
  </vivo:Company>
  ...
</rdf:RDF>
```

Notes:

1. The return is RDF/XML containing the first 100 triples of the result set. We see a company being described with a series of assertions.
2. The return contains a description of the returned data. There is a void:Dataset description containing information about the query, its size (5,113,025 triples), and that 100 are included. The description also contains URL than can be used to navigate to the next page and first page in the result set.

10.2.9.4.1 IRI Patterns

The following IRI patterns are valid for making TPF requests from curl or from software (see below).

```
http://example.org/tpf/core?subject={subject}&predicate={predicate}&object={object}
http://example.org/tpf/core?s={subject}&p={predicate}&o={object}
```

For example:

```
curl http://openvivo.org/tpf/core?subject=http://openvivo.org/a/
orcid0000-0002-1304-8447
```

Returns an RD/XML document containing the first 100 triples regarding the specified subject.

10.2.9.4.2 Headers

Headers can be used to specify the output format.

```
curl -H "Accept: application/n-triples; charset=utf-8" http://openvivo.org/tpf/core?
subject=http://openvivo.org/a/orcid0000-0002-1304-8447
```

Which returns 166 triples as of this writing.

10.2.9.5 Programmatic Access

Programmatic access follows the same approach as curl. Issue an HTTP request for the specified data. Here's a simple Javascript JQuery Ajax code fragment making a TPF request. The code expects a siteUri, a subjectUri and a predicateUri, and returns all triples with the specified subjectUri and predicateUri. A dataFilterFunction is called on the return prior to the successFunction being called.

```
        $.ajax({
            headers: {Accept : "application/n-triples; charset=utf-8"},
            url: siteUri,
            data: {subject: subjectUri, predicate: predicateUri, object: "", page:
"1"},
            dataFilter: function(data) { dataFilterFunction(data); },
            success: function(data) { successFunction(data); }
            });
```

10.2.9.6 References

1. Linked Data Fragments In-depth http://linkeddatafragments.org/in-depth/
2. Triple Pattern Fragments specification http://www.hydra-cg.com/spec/latest/triple-pattern-fragments/
3. Verborgh, R. et al. Triple Pattern Fragments: A low cost knowledge graph interface for the web https://www.sciencedirect.com/science/article/pii/S1570826816000214?via%3Dihub
4. Verborgh, R. The Future is Federated. Invited presentation at 2016 VIVO Conference, Denver, Colorado. http://openvivo.org/display/doi10.6084/m9.figshare.3680310
5. LinkedDataFragments Server. Github. https://github.com/LinkedDataFragments/Server.Java

10.3 Application RDF Files

When VIVO is started for the first time, its empty triple stores ("triple sources") are initialized with data from RDF files deployed to the rdf subdirectory of the VIVO home directory. These RDF files are organized into a number of subdirectories in order to separate different types of data and to identify which parts of the data will be updated and maintained via edits to the corresponding triple store and which will be modified by subsequent changes to these RDF files.

10.3.1 ABox, TBox and Configuration Data

The top-level subdirectories in the rdf directory can be organized into three general categories:

1. Data about the structure of ontologies (the “terminological box” or TBox)

a. Example: foaf:Person is a class and it is a subclass of foaf:Agent.
b. Directory:

1. `/tbox`

1. Data about specific named “individuals” and their relationships (the “assertion box” or ABox)

a. Example: Botswana is a country and it is located on the continent of Africa.
b. Directory:

1. `/abox`

1. Data for configuring the behavior of the Vitro application

a. Examples:

i. There is a class group called “People” and it has a rank of 1 for sorting in a list of class groups.
ii. The SPARQL Update API is restricted to the root user.
iii. The “relatedBy” property, when the object is of class vivo:Authorship, should be rendered on a profile page with the heading “selected publications.”
iv. A custom form should be used when editing a certain configuration item.
v. display:NavigationElement is a class used in the configuration data.

a. Directories:

1. `/applicationMetadata`
2. `/auth`
3. `/display`
4. `/displayDisplay`
5. `/displayTbox`

Why keep these separate? In a typical VIVO, the data about specific individuals (the ABox) is much larger than the ontology structure data (the TBox) or the data used for application configuration. The latter two are accessed repeatedly and benefit from being cached in memory. By keeping these areas of concerns separate it is easy to load the ontology definitions and the configuration data into memory without performing a complicated process of extracting them from the rest of the data in the triple store.

10.3.2 Content and Configuration Triple Sources

VIVO uses two triple sources as configured in applicationSetup.n3 in the VIVO home directory. Data from these directories is stored in the content triple source:

> `/abox`
> `/tbox`
> `/applicationMetadata`

Data from these directories is stored in the configuration triple source:

> `/auth`
> `/display`
> `/displayDisplay`
> `/displayTbox`

10.3.3 Firsttime, Everytime and Filegraph

Within the directories that delineate the above categories of RDF data, there are further divisions that specify where the respective data should be stored and updated in the future.

10.3.3.1 Firsttime

RDF files in "firsttime" directories are loaded when the triple store is initialized or upgraded: that is, the very first time VIVO is started with an empty triple store, or the first time a new version of VIVO is started with an existing triple store if the new version of VIVO contains an upgraded version of the ontology.

"Firsttime" data is expected to be modifiable by end users or system administrators through VIVO's Web interface -- either using interactive editing forms or by loading or removing RDF data via the Site Admin page. Firsttime files are the standard way of distributing the parts of ontologies that do not have semantic meaning (do not affect the inferences made by reasoners). Prior to version 1.10, these annotations also included the human-readable labels applied to classes and properties.

10.3.3.1.1 First time a new VIVO is started

When a new VIVO is started, each RDF file in each firsttime directory is loaded in its entirety into one of the application's triple stores. The data are stored in graphs that can be written to by VIVO's GUI editing interface or modified by the Add/Remove RDF feature.

10.3.3.1.2 First time a new version of VIVO is started with an upgraded ontology

When a new version of VIVO is started with a new version of the ontology, the upgrade process compares the previous version of the "firsttime" files against the current state of the triple store. For a given subject and predicate, if the value in the triple store still matches the value from the previous version of firsttime -- that is, end users have not made a change to this data element -- then any changes found in the current version of firsttime will be propagated to the triple store.

Example: how VIVO applies RDF updates when changes have been made using the user interface

1. Version 0.1 rdf/tbox/firsttime/vitroAnnotations.n3 contains the triples 'foaf:Person vitro:displayRankAnnot "-1"^^xsd:int and 'foaf:Organization rdfs:label vitro:displayRankAnnot "-1"^^xsd:int'.
2. VIVO is installed with a new empty triple store
3. Triple from step 1 is loaded into the triple store.
4. VIVO is restarted any number of times; the initialTBoxAnnotations.n3 file is no longer consulted.
5. A user edits the display rank in the VIVO UI for the foaf:Person class, changing it to "9".
6. Version 0.2 is installed, where the vitroAnnotations.n3 file now contains the triples 'foaf:Person vitro:displayRankAnnot "5"^^xsd:int and 'foaf:Organization vitro:displayRankAnnot "2"^^xsd:int'. The new code installation continues to use the existing triple store with all of its data intact.
7. VIVO is restarted with the new version.

8. VIVO’s upgrade process consults the current and previous vitroAnnotations.n3 files and discovers that the foaf:Person class has a new display rank.
9. It checks the triple store and sees that “9” differs from the previous value of “-1” -- that is, someone has edited it.
10. It keeps the value that someone has entered and ignores the new value of “5” found in the file.
11. The upgrade process discovers that foaf:Organization has a new display rank.
12. It checks the triple store and sees that the display rank is “-1” and matches the value from the previous version of vitroAnnotations.n3 -- that is, no one has edited it.
13. It removes the value “-1” from the triple store and adds the new value of “2”.

10.3.3.2 Everytime

“Everytime” files are used for configuration data RDF files that are loaded each time VIVO starts and whose statements are added to a temporary in-memory RDF model. It is not possible to identify which statement came from which file. Data in everytime files are not intended to be editable by end users using VIVO’s GUI interface. Any further updates to the data are to be made on the filesystem, followed by a restart of VIVO.

10.3.3.3 Filegraph

“Filegraph” files are loaded each time VIVO starts, and the contents of each file are loaded into a separate graph in the triple store. Thus, it is possible to query the triple store and discover from which file a given statement was loaded -- or, for example, to output all of the statements that came from a given ontology. Triples in the filegraph files are not deletable by end users using VIVO’s GUI interface. Any further updates to the data that require removal of the original triples are to be made on the filesystem, followed by a restart of VIVO. Files in the filegraph directory are the standard method of distributing all parts of existing ontologies that have semantic meaning (affect the inferences made by reasoners). For example, the class hierarchy of the VIVO ontology is not intended to be modified by end users because it would make their data incompatible with that of other VIVO users.

When a filegraph file is modified and VIVO is restarted, a background recomputation of all of VIVO’s inferred triples will be performed. This is done in order to take into account the logical consequences of any new ontology axioms that might have been added via the modification. (Note that if you shut down VIVO while this background process is running, it will not automatically resume but may be started again manually via the “Recompute inferences” link on the Site Admin page.)

10.3.3.3.1 Example

When VIVO is started, the contents of the file at rdf/tbox/filegraph/vivo.owl are compared to the contents of the graph http://vitro.mannlib.cornell.edu/filegraph/tbox/vivo.owl in the triple store. (Specifically, it is tested to see if the two graphs are isomorphic or have the same shape.)

If the contents differ, the graph in the triple store is emptied and the statements from the file are loaded into it. After a reload, a complete recomputation of VIVO’s inferences is triggered.

10.4 Architecture

10.4.1 Overview

VIVO is an enterprise class software system relying on numerous open source software components. Fundamentally, VIVO relies on Vitro (see below). VIVO adds a collection of ontologies (see Ontology Reference (see page 446)) to represent data about scholarship.

10.4.2 Vitro

Vitro is an open source, general purpose, semantic web engine. It is the application development platform underlying VIVO. Vitro has no domain knowledge. Given ontologies regarding a domain, Vitro supports the editing of the ontology, creation of individuals, management of individuals on "pages" which it generates, organization of individuals into "class groups," indexing, search, faceted browsing, query, import, and export. Vitro has been used to manage collections of clinical trials, spaceships, library catalogs, datasets, and many more.

VIVO is Vitro with an ontology for representing scholarship, and a set of displays and visualizations that support the use of data for expert finding, team building, assessment, and other VIVO use cases.

Vitro can be built and operated independently of VIVO. VIVO is completely dependent on Vitro.

10.4.3 VIVO

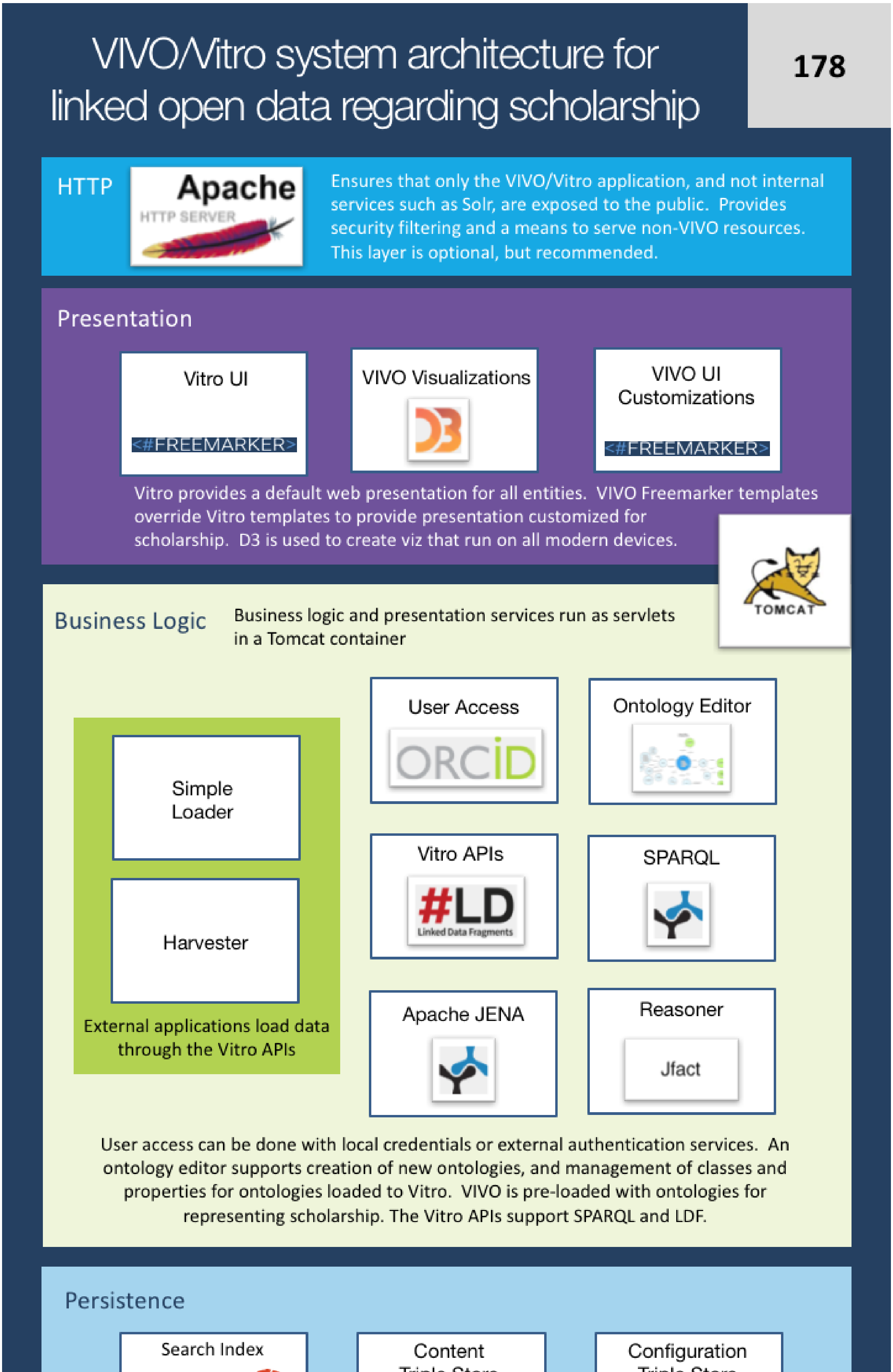
VIVO is a customized Vitro. The table below shows how VIVO compares to Vitro.

	Vitro	**VIVO**
Purpose	General-purpose tool for working with Semantic Data	Specialized tool for Research Networking
Ontology	No ontology	Includes an ontology (VIVO-ISF) for Research Networking
Theme	Minimal theme	Elaborate theme, display and editing are customized for the ontology
Display Rules	Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page
Form editing	Default editing forms	Editing is customized to the ontology

	Vitro	**VIVO**
Search Index	Default search index	Search index contains additional fields specific to VIVO
Functionality	Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.

10.4.4 Component View

VIVO, with Vitro, as "made" out of components, including other open source software components. The figure below shows the various software components that are used in a VIVO/Vitro system.



VIVO/Vitro system architecture for linked open data regarding scholarship

HTTP

Apache
HTTP SERVER

Ensures that only the VIVO/Vitro application, and not internal services such as Solr, are exposed to the public. Provides security filtering and a means to serve non-VIVO resources. This layer is optional, but recommended.

Presentation

Vitro UI
<#FREEMARKER>
VIVO Visualizations
VIVO UI Customizations
<#FREEMARKER>

Vitro provides a default web presentation for all entities. VIVO Freemarker templates override Vitro templates to provide presentation customized for scholarship. D3 is used to create viz that run on all modern devices.

Business Logic

Business logic and presentation services run as servlets in a Tomcat container

Simple Loader
Harvester
User Access
ORCID
Ontology Editor
Vitro APIs
#LD
Linked Data Fragments
SPARQL
Apache JENA
Reasoner
Jfact

External applications load data through the Vitro APIs

User access can be done with local credentials or external authentication services. An ontology editor supports creation of new ontologies, and management of classes and properties for ontologies loaded to Vitro. VIVO is pre-loaded with ontologies for representing scholarship. The Vitro APIs support SPARQL and LDF.

Persistence

Search Index
Content
Configuration

10.4.5 Additional Resources

- Vitro (see page 398)
- VIVO and Vitro (see page 398)
- Software Architecture Overview (see page 399)
- The StartupManager (see page 403)
- VIVO Data Models (see page 406)
- VIVO and the Solr search engine (see page 418)
- Image storage (see page 425)

10.4.6 Vitro

Vitro is an open source, general purpose, semantic web engine. It is the application development platform underlying VIVO. Vitro has no domain knowledge. Given ontologies regarding a domain, Vitro supports the editing of the ontology, creation of individuals, management of individuals on "pages" which it generates, organization of individuals into "class groups," indexing, search, faceted browsing, query, import, and export. Vitro has been used to manage collections of clinical trials, spaceships, library catalogs, datasets, and many more.

VIVO is Vitro with an ontology for representing scholarship, and a set of displays and visualizations that support the use of data for expert finding, team building, assessment, and other VIVO use cases.

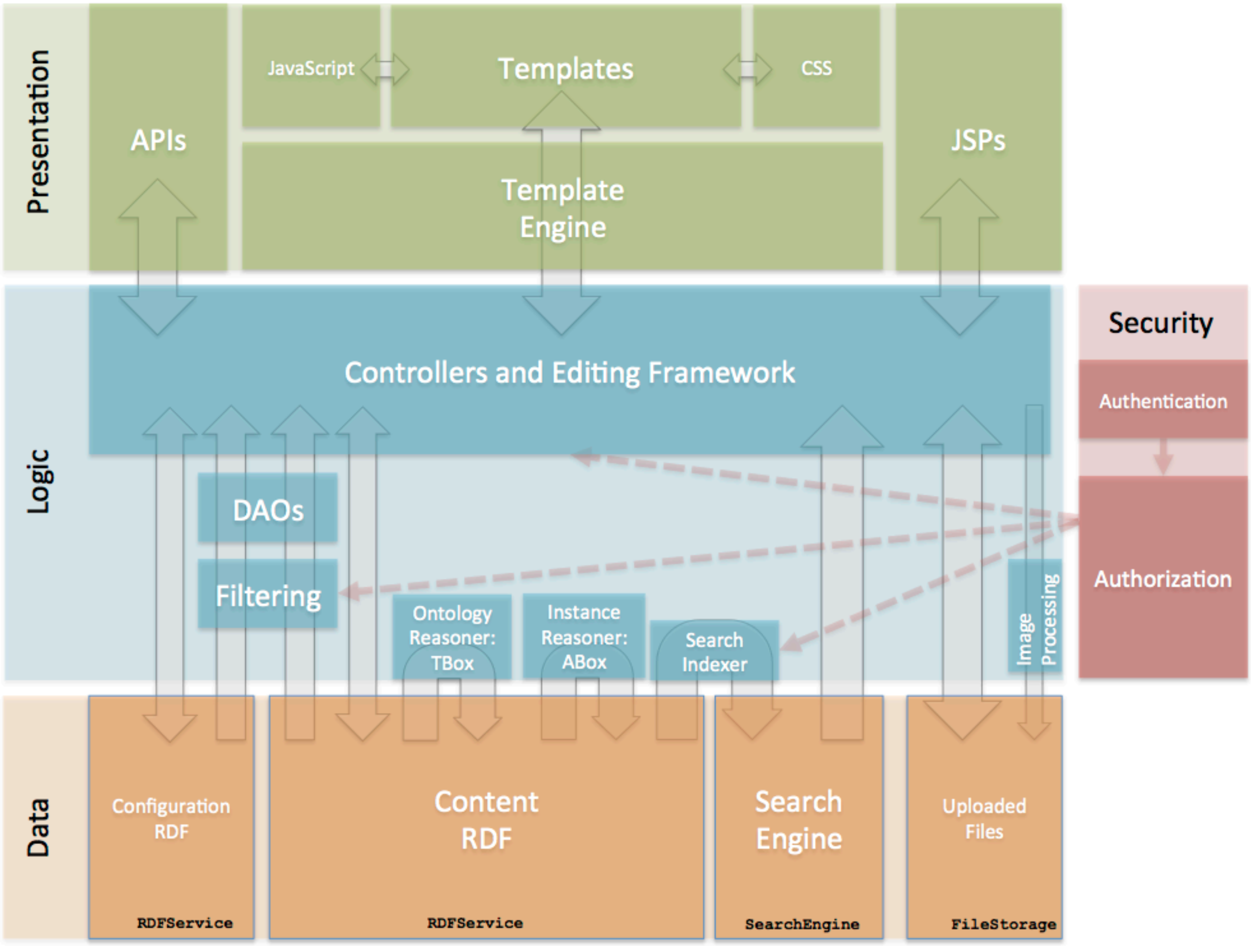
10.4.7 VIVO and Vitro

VIVO itself is a customization of a more generic product called Vitro. Here is how VIVO has been customized from Vitro.

	Vitro	**VIVO**
Purpose	General-purpose tool for working with Semantic Data	Specialized tool for Research Networking
Ontology	No ontology	Includes an ontology (VIVO-ISF) for Research Networking
Theme	Minimal theme	Elaborate theme, display and editing are customized for the ontology
Display Rules	Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page

	Vitro	VIVO
Form editing	Default editing forms	Editing is customized to the ontology
Search Index	Default search index	Search index contains additional fields specific to VIVO
Functionality	Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.

10.4.8 Software Architecture Overview


Presentation
APIs
JavaScript
Templates
CSS
JSPs
Template Engine
Logic
Controllers and Editing Framework
Security
Authentication
Authorization
DAOs
Filtering
Ontology Reasoner: TBox
Instance Reasoner: ABox
Search Indexer
Image Processing
Data
Configuration RDF
RDFService
Content RDF
RDFService
Search Engine
SearchEngine
Uploaded Files
FileStorage

2 Components of VIVO

10.4.8.1 Data

VIVO has four data stores. When copying, backing up, or restoring a VIVO installation, all four data stores should be considered.

10.4.8.1.1 Content RDF

This is where most of VIVO's information is stored. Names of individuals, relationships between individuals, types of individuals (for example, Person or Organization), are all stored in the Content RDF

Content RDF uses a triple-store or other SPARQL endpoint. Usually, the triple-store is a Jena SDB implementation, with a MySQL database.

The interface is specified by `RDFService.java`.

10.4.8.1.2 Configuration RDF

This is where VIVO's parameters are stored, like which templates are used to display what types of data. Other parameters describe how the custom editing screens are applied to complex data structures. The Configuration RDF is also the storage for VIVO's user accounts.

Configuration RDF uses a triple-store or other SPARQL endpoint. The triple-store is a Jena TDB implementation, with files kept in the home directory of the VIVO application.

The interface is specified by `RDFService.java`.

10.4.8.1.3 Search Engine

In theory, all of the search operations in VIVO could be performed using SPARQL queries against the RDF. In practice, however, a dedicated search engine gives a much faster response. The search engine is available to VIVO's users, to assist in finding pages of interest. The search engine is also used internally, to provide prompt response to requests for auto-completion, indexes, counts, and other data.

The search engine permits queries that yield faceted results, for a more successful search. Usually, it is implemented with a Solr web application. By default, Solr is installed in the same web server as VIVO. However, it is easy to move Solr to a different web server, to improve performance.

The interface is specified by `SearchEngine.java`

10.4.8.1.4 Uploaded Files

VIVO allows individuals to upload images for their profile pages. VIVO also generates a thumbnail image for more compact display. These images are kept in the Uploaded Files storage. Each file is assigned a URI, so it can be distinguished from other files of the same name. Currently this is only used for images, but VIVO could be customized to store other types of files here as well.

The default implementation uses a storage system similar to PairTree[158].

---

[158] https://wiki.ucop.edu/display/Curation/PairTree

The interface is specified by `FileStorage.java`

10.4.8.2 Logic

VIVO adds a layer of "business logic" to the data storage. It uses inference to add to the data. It applies policies to determine which users are authorized to see which pieces of data.

10.4.8.2.1 Controllers and Editing Framework

The controllers contain the top-level logic, determining how to respond to web requests. This includes fetching data, making decisions based on that data, and displaying the results.

The Editing Framework provides the user with the tools needed to edit the RDF data. In some cases, a simple default screen will suffice. For more elaborate data structures, the Editing Framework creates related groups of data objects, and enforces the relationships among them.

10.4.8.2.2 DAOs

The DAOs, or Data Access Objects, form a layer of secondary logic. They provide a large number of utility subroutines, to take the repetitive processing tasks away from the Controllers and Editing Framework.

The DAOs also provide a framework for the filtering layer.

There are a large number of interfaces that define the DAO layer.

10.4.8.2.3 Filtering

Data within VIVO can be public or private, or shades of gray. The Filtering layer works with the Authentication system to determine which pieces of data may be displayed to a particular user.

The Filtering layer means that the Controllers don't need to include logic for this sort of decision. The Controller asks the Filtered DAOs for data, and receives as much data as the current user is authorized to see.

The interface is specified by `VitroFilters.java`

10.4.8.2.4 Ontology Reasoners: ABox and TBox

One of the principal strengths of RDF is that we can infer additional data from the data at hand. However, the logic involved can be complicated and time-consuming.

Currently VIVO applies two different reasoners to the Content RDF. The TBOX - or Ontology models - are small enough that extensive reasoning can be applied. Currently, the JFact reasoner is used. Applying that same level of inference to the ABOX - or Assertions models - would take a prohibitive amount of time. VIVO uses its own reasoner for this, applying only those logical inferences that VIVO requires to function.

10.4.8.2.5 Search Indexer

The Search Indexer reacts to changes in the Content RDF, updating the search index to reflect those changes. Several types of logic are employed to determine which individuals are affected by the RDF changes, and how to build the search records for those individuals. Sometimes a single change requires that several search records be rebuilt.

10.4.8.2.6 Image Processor

When images are uploaded through the GUI, the Image Processor creates a thumbnail image, cropped and sized as the user requests. Currently, the image processor is based on the Java Advanced Imaging library.

10.4.8.3 Presentation

The presentation layer is where the web pages of VIVO are created. Most of the web pages are created using the Freemarker template engine. However, a number of pages are still created by JSPs.

10.4.8.3.1 Template Engine and Templates

VIVO uses the Freemarker Template Engine to construct the HTML for its web pages. The templates describe the format and structure of the pages, and the template engine inserts relevant data each time the template is used.

10.4.8.3.2 JavaScript

VIVO relies heavily on JQuery to create a rich and responsive user interface. Other scripts are used also.

10.4.8.3.3 CSS

Like any web application, VIVO uses CSS files to produce a consistent style across the user interface.

10.4.8.3.4 JSPs

Early releases of VIVO were built almost entirely using JSPs. The change to Freemarker was an attempt to insure better separation between the Logic layer and the Presentation layer.

Some JSPs are still used in VIVO. In general these are restricted to administrative pages, including advanced data manipulation and "back-end editing".

10.4.8.3.5 APIs

VIVO supports a collection APIs for importing and exporting data. The APIs have no presentation layer, per se. The format of their responses is determined by the nature of the request, and usually does not involve HTML. Responses to API requests are constructed entirely by the Controllers.

10.4.8.4 Security

The security system determines what data a user may see, what data they may modify, and what functions they may perform.

10.4.8.4.1 Authentication

VIVO includes its own authentication system, including user accounts with email addresses as identifiers and passwords as credentials. VIVO can be configured to use an external authentication system also. In this case, VIVO still maintains a user account for each user, but no passwords are stored. If the external authentication system asserts that a user has properly logged in, VIVO accepts that assertion.

10.4.8.4.2 Authorization

The Authorization system relies on a list of Policy objects to determine what a user may do. Before the Controllers or the Editing Framework or the Filtering layer take any action, they consult the Policy list to determine whether that action is authorized for the current user.

This very flexible set of Policies permits VIVO to classify some data as public or private, while other data is private except to the user who owns it.

10.4.9 The StartupManager

10.4.9.1 Overview

Like most Java Enterprise applications, Vitro servlets rely on the ServletContext to hold object that they will need to use when servicing requests. These objects are created by ServletContextListeners, which are run by the StartupManager.

The StartupManager creates instances of the listeners and runs them, accumulating information about their running in the StartupStatus.

As each listener runs, it may add messages to the StartupStatus. Each message will have a severity level associated with it:

- FATAL – The listener encountered a problem. Perhaps the application was configured incorrectly, or perhaps the system utilities are not performing as intended. The problem is severe enough that the application will not run. The message describes the problem, with suggestions on how to fix it.
- WARNING – The listener encountered a problem, but the problem will not prevent the application from running. The message describes the problem, and tells what parts of the application will be affected, with suggestions on how to fix the problem.
- INFO – No problem is indicated. The message contains information that may be helpful in monitoring the application.

If a FATAL status is recorded, the StartupManager will not execute any additional listeners. Access to the application will be blocked, and any attempt to access the application will display the StartupStatus in an error page.

If a WARNING status is recorded, the StartupManager continues as normal. Access to the application will be blocked one time, to display the StartupStatus. In subsequent requests, the application will respond normally.

When logged in, an administrator may view the StartupStatus from a link on the Site Admin page.

10.4.9.2 Specifying context listeners

In any Java Enterprise application, developers can specify context listeners in the deployment descriptor (web.xml). These listeners that will be activated when the application starts and when it shuts down.

In Vitro, the only listener in web.xml is the StartupManager. Here is the relevant section of Vitro’s web.xml:

```
 <!--
    StartupManager instantiates and runs the listeners from
startup_listeners.txt
    All ServletContextListeners should be listed there, not here.
 -->
 <listener>
   <listener-class>edu.cornell.mannlib.vitro.webapp.startup.StartupManager</
listener-class>
 </listener>
```

Vitro contains a list of startup listeners in a file at Vitro (see page 398)/webapp/config/startup_listeners.txt. This file is simple text with each line containing the fully-qualified class name of a startup listener. Blank lines are ignored, as are comment lines – lines that begin with a “hash” character. Here is a portion of that file:

```
 #
 # ServletContextListeners for Vitro,
 # to be instantiated and run by the StartupManager.
 #

 edu.cornell.mannlib.vitro.webapp.config.ConfigurationPropertiesSetup

 edu.cornell.mannlib.vitro.webapp.config.RevisionInfoSetup

 edu.cornell.mannlib.vitro.webapp.email.FreemarkerEmailFactory$Setup

 # DefaultThemeSetup needs to run before the JenaDataSourceSetup to allow
creation
 # of default portal and tab
 edu.cornell.mannlib.vitro.webapp.servlet.setup.DefaultThemeSetup
```

10.4.9.3 Writing context listeners

Each listener must implement the ServletContextListener interface, and must have a zero-argument constructor.

When Vitro starts, the StartupManager will call contextInitialized() in each listener, in the order that they appear in the file. The listener can call methods on StartupStatus to record messages. If the listener is successful, it should record one or more INFO messages that provide a brief description of what it has done. If a problem is detected, the listener may record WARNING messages or ERROR messages, depending on the severity of the problem. The listener may also throw a RuntimeException from contextInitialized(), which the StartupManager will treat like an ERROR.

Here is an example of a basic listener. When contextInitialized() is called, the listener will perform some setup. If there is no problem, a call to StartupStatus.info() reports some basic information about the listener's actions. If a problem is found, a call to StartupStatus.warning() describes the nature of the problem (by reporting the exception) and how this problem will affect the application.

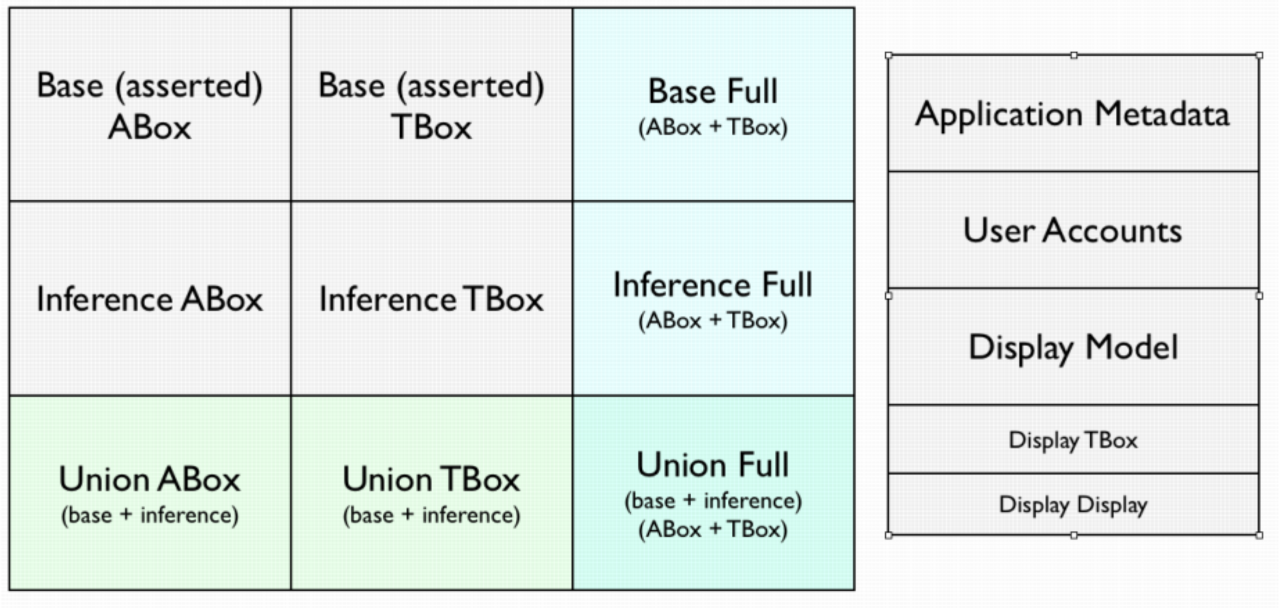
```
 public static class Setup implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext ctx = sce.getServletContext();
        StartupStatus ss = StartupStatus.getBean(ctx);

        try {
            FreemarkerEmailFactory factory = new
FreemarkerEmailFactory(ctx);
            ctx.setAttribute(ATTRIBUTE_NAME, factory);

            if (factory.isConfigured()) {
                ss.info(this, "The system is configured to "
                        + "send mail to users.");
            } else {
                ss.info(this, "Configuration parameters are missing: "
                        + "the system will not send mail to users.");
            }
        } catch (Exception e) {
            ss.warning(this,
                    "Failed to initialize FreemarkerEmailFactory. "
                            + "The system will not be able to send email "
                            + "to users.", e);
        }
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        sce.getServletContext().removeAttribute(ATTRIBUTE_NAME);
    }
}
```

Note that the StartupManager treats ServletContextListeners just like you would expect from reading the Servlet 2.4 specification:

- Only one instance of the listener is created per JVM.
- The contextInitialized() method is called once when the system is starting.
- The contextDestroyed() method is called on that same instance when the system shuts down.

10.4.10 VIVO Data Models

10.4.10.1 Concepts


Base (asserted) ABox
Base (asserted) TBox
Base Full (ABox + TBox)
Inference ABox
Inference TBox
Inference Full (ABox + TBox)
Union ABox (base + inference)
Union TBox (base + inference)
Union Full (base + inference) (ABox + TBox)
Application Metadata
User Accounts
Display Model
Display TBox
Display Display

Frequently, we talk about "the data model" in VIVO. But this is an over-simplification which can be useful at times, but misleading at other times. In fact, VIVO contains a matrix of data models and sub-models, graphs, datasets and other constructs.

It might be more accurate to talk about the union of these data models as "the knowlege base". However, the terminology of "the data model" is firmly entrenched.

Beginning in VIVO release 1.6, we are attempting to simplify this complex collection of models, and to produce a unified access layer. This is a work in progress. Regardless of how clean the design might eventually become, this will remain an area with complex requirements which cannot be satisfied by simplistic solutions.

10.4.10.1.1 Divisions in the knowledge base

Depending on what you want to do with the data, it can be useful to sub-divide it by one or more of the following criteria:

10.4.10.1.1.1 Types of statements

An RDF model is often divided into ABox (assertions) and TBox (terminology). In RDF, there is no technical distinction between TBox and ABox data. They are stored separately because they are used for different purposes. The combination of the two is informally called the Full model.

	Data type	**Example data**
TBox	"Terminological data" Defines classes, properties, and relationships in your ontology.	
ABox	"Assertion data" Enumerates the individual instances of your classes and describes them.	

TBox

```
foaf:Person
   a owl:Class ;
   rdfs:subClassOf owl:Thing ;
   rdfs:label "Person"@en .
ex:preferredName
   a owl:DatatypeProperty ;
   rdfs:subPropertyOf skos:prefLabel,
                      foaf:name,
                      rdfs:label ;
   rdfs:domain foaf:Person ;
   rdfs:label "preferred name"@en .
```

ABox

```
local:tobyink
   a foaf:Person ;
   ex:preferredName "Toby Inkster" .
```

	Data type	Example data
Full	The TBox and the ABox together, treated as a single model. For example, when you use the RDF tools to remove statements, you want them removed regardless of whether they are found in the TBox or the ABox.	

10.4.10.1.1.2 Source of statements

An RDF model can also be divided into Assertions and Inferences. The combination of the two is informally called the Union.

Statement type	Meaning	Example data
Assertions	Statements that you explicitly add to the model, either through setup, ingest, or editing.	`local:tobyink`[159] `rdfs:type`[160] `core:FacultyMember .`

Inferences	Statements that the semantic reasoner adds to the model, by reasoning about the assertions, or about other inferences.	`local:tobyink rdfs:type foaf:Person .`
		`local:tobyink rdfs:type foaf:Agent .`
		`local:tobyink rdfs:type owl:Thing .`

Union	The combination of Assertions and Inferences. For most purposes, this is the desired model. You want to know what statements are available, without regard to whether they were asserted or inferred.	

---

[159] http://localtobyink

[160] http://rdfstype

10.4.10.1.1.3 "Content" vs. "Configuration"

We sometimes distinguish between the data that VIVO is serving (Content) and the data that VIVO itself uses (Configuration). The Content is available for display, for searching, for serving as Linked Open Data. The Configuration controls how the content is displayed, who can access the data, and what VIVO itself looks like.

Model type	Purpose	Examples
Configuration	Data about the VIVO application itself.	Application parameters User Accounts Display options
Content	The payload - the data that VIVO is intended to distribute.	People data Publications data Grant data etc.

10.4.10.1.1.4 Model scope

The knowledge base exists for as long as VIVO is running. However, subsets or facets of the knowledge base are often used to satisfy a particular HTTP request, or through the length of a VIVO session for a particular user. These subsets are created dynamically from the full knowledge base, used for as long as they are useful, and then discarded.

Scope	Purpose	Example	Discarded when...
Application (Servlet Context)	Created for the life of VIVO.		Never discarded.
Session	Created for a particular logged-in user	Data that is filtered by what the user is permitted to view.	When the user logs out, or the session times out.
Request	Created for a single HTTP request	Data that is organized by the languages that are preferred by the browser.	When the individual request has been satisfied.

At present, the Session lifespan is almost never used. However, potential use cases do exist for it.

The Request lifespan is used extensively, since it provides a convenient way to manage database connections and minimize contention for resources.

10.4.10.1.2 Purpose vs. scope

It is tempting to think of the models of the Servlet Context as equivalent to the unfiltered models of the Request. They may even represent the very same data. However, they have different scope, which makes them very different in practice.

The unfiltered models in the Request go out of scope when the Request has been satisfied. The resources required by these models have short lifetimes and are very easily managed. The models of the Servlet Context never go out of scope until VIVO is shut down. It is difficult to reclaim resources such as database connections or processor memory from these models.

10.4.10.1.3 Filtering

To enable language filtering and internationalization (see page 129)of the user interface (translation of data labels and contents, localization, etc.), you should update the **runtime.properties** file:

```
RDFService.languageFilter = true
```

and then edit the list of needed languages:

```
languages.selectableLocales = en_US, es, de_DE, fr_CA, pt_BR
```

10.4.10.2 The Data Models

This is a summary of the data models:

The basic content	Base ABox, Base TBox, Inferred ABox, Inferred TBox	Named graphs from the RDF Service (optionally with sub-graphs).
Views of the content	Base Full, Inferred Full, Union ABox, Union TBox, Union Full	Views of the 4 basic content graphs in different combinations.
The configuration	Application Metadata, User Accounts, Display Model, Display TBox, DisplayDisplay	Named graphs from the application datasource.

10.4.10.3 Increasing complexity

The structure of the data models has grown as VIVO has developed. New models, new structures, and new means of accessing the data have been added as required by the growing code. The resulting data layer has grown more complex and more error-prone.

The `RDFService` interface, increases the flexibility of data sources, and promises to allow a more unified view of the knowledge base. However, the transition to `RDFService` is not complete, and so this adds another layer of complexity to the data issues. New structures have been added, but none removed.

10.4.10.3.1 Beyond the models

There is an incredible variety of ways to access all of these models. Some of this variety is because the models are accessed in different ways for different purposes. Additional variety stems from the evolution of VIVO in which new mechanisms were introduced without taking the time and effort to phase out older mechanisms.

Here are some of the ways for accessing data models:

10.4.10.3.1.1 Attributes on Context, Session, or Request

Previously, it was common to assign a model to the ServletContext, to the HTTP Session, or to the HttpSessionRequest like this:

```
OntModel ontModel = (OntModel) getServletContext().getAttribute("jenaOntModel");

Object sessionOntModel = request.getSession().getAttribute("jenaOntModel");
```

Occasionally, conditional code was inserted, to retrieve a model from the Request if available, and to fall back to the Session or the Context as necessary. Such code was sporadic, and inconsistent. This sort of model juggling also involved inversions of logic, with some code acting so a model in the Request would override one in the Session, while other code would prioritize the Session model over the one in the Request. For example:

```
public OntModel getDisplayModel(){
    if( _req.getAttribute("displayOntModel") != null ){
        return (OntModel) _req.getAttribute(DISPLAY_ONT_MODEL);
    } else {
        HttpSession session = _req.getSession(false);
        if( session != null ){
            if( session.getAttribute(DISPLAY_ONT_MODEL) != null ){
                return (OntModel) session.getAttribute(DISPLAY_ONT_MODEL);
            }else{
                if( session.getServletContext().getAttribute(DISPLAY_ONT_MODEL) != 
null){
                    return
 (OntModel)session.getServletContext().getAttribute(DISPLAY_ONT_MODEL);
                }
                }
            }
        }
        log.error("No display model could be found.");
        return null;
    }
```

This mechanism has been removed in 1.6, being subsumed into the `ModelAccess` class (see below). Now, the `ModelAccess` attributes on Request, Session and Context are managed using code that is private to `ModelAccess` itself. Similarly, the code which gives priority to a Request model over a Session model is uniformly implemented across the models.

It remains to be seen whether this uniformity can satisfy the various needs of the application. If not, at least the changes can all be made within a single point of access.

10.4.10.3.1.2 The DAO layer

This mechanism is pervasive through the code, and remains quite useful. In it, a `WebappDaoFactory` is created, with access to particular data models. This factory then can be used to create DAO objects which satisfy interfaces like `IndividualDao`, `OntologyDAO`, or `UserAccountsDAO`. Each of these object implements a collection of convenience methods which are used to manipulate the backing data models.

Because the factory and each of the DAOs is an interface, alternative implementations can be written which provide

- Optimization for Jena RDB models
- Optimization for Jena SDB models
- Filtering of restricted data
- and more...

Initially, the `WebappDaoFactory` may have been used only with the full Union model. But what if you want to use these DAOs only against asserted triples? Or only against the ABox? This led to the `OntModelSelector`.

10.4.10.3.1.3 OntModelSelectors

An `OntModelSelector` provides a way to collect a group of Models and construct a `WebappDaoFactory`. With slots for ABox, TBox, and Full model, an `OntModelSelector` could provide a consistent view on assertions, or on inferences, or on the union. The `OntModelSelector` also holds references to a display model, an application metadata model, and a user accounts model, but these are more for convenience than flexibility.

Prior to release 1.6, `OntModelSelectors`, like `OntModel`s, were stored in attributes of the Context, Session, and Request. They have been subsumed into the `ModelAccess` class.

Further, the semantics of the "standard" `OntModelSelectors` have changed, so they only act as facades before the Models store in `ModelAccess`. In this way, if we make this call:

```
ModelAccess.on(session).setOntModel(ModelID.BASE_ABOX, someWeirdModel)
```

Then both of the following calls would return the same model:

```
ModelAccess.on(session).getOntModel(ModelID.BASE_ABOX);
ModelAccess.on(session).getBaseOntModelSelector().getABoxModel();
```

Again, this is a change in the semantics of OntModelSelectors. It insures a consistent representation of `OntModel` s across `OntModelSelector` s, but it is certainly possible that existing code relies on an inconsistent model instead.

10.4.10.3.1.4 The RDF Service

Interface for API to write, read, and update Vitro's RDF store, with support to allow listening, logging and auditing. Moreover, it is an intefrace for API to perform a SPARQL select query against the knowledge base. The query may have an embedded graph identifier. If the query does not contain a graph identifier the query is executed against the union of all named and unnamed graphs in the store.

At the end, implementation of this interface should enable serialization of the contents of the named graph to the supplied OutputStream, in N-Triples format.

10.4.10.3.1.5 Model makers and Model sources

10.4.10.4 The ModelAccess class

The root access point for the RDF data structures: RDFServices, Datasets, ModelMakers, OntModels, OntModelSelectors and WebappDaoFactories.
Enables getting a long-term data structure by accessing from the context.

```
ModelAccess.on(ctx).getRDFService(CONFIGURATION);
```

Moreover it enables getting a short-term data structure by accessing from the request.

```
ModelAccess.on(req).getOntModel(ModelNames.DISPLAY);
```

The elaborate structure of options enums allows us to specify method signatures like this on RequestModelAccess:

```
getOntModelSelector(OntModelSelectorOption... options);
```

Which can be invoked in any of these ways:

```
ModelAccess.on(req).getOntModelSelector();
ModelAccess.on(req).getOntModelSelector(LANGUAGE_NEUTRAL);
ModelAccess.on(req).getOntModelSelector(INFERENCES_ONLY);
ModelAccess.on(req).getOntModelSelector(ASSERTIONS_ONLY, LANGUAGE_NEUTRAL);
```

The compiler insures that only appropriate options are specified. However, if conflicting options are supplied, it will only be caught at runtime.

10.4.10.5 Initializing the Models

When VIVO starts up, `OntModel` objects are created to represent the various data models. The configuration models are created from the datasource connection, usually to a MySQL database. The content models are created using the new RDFService layer. By default this also uses the datasource connection, but it can be configured to use any SPARQL endpoint for its data.

Some of the smaller models are "memory-mapped" for faster access. This means that they are loaded entirely into memory at startup. Any changes made to the memory image will be replicated in the original model.

The data in each model persists in the application datasource (usually a MySQL database), or in the RDFService. Also, data from disk files may be loaded into the models. This may occur:

- the first time that VIVO starts up,
- if a model is found to be empty,
- every time that VIVO starts up.

depending on the particular model.

10.4.10.5.1 Where are the RDF files?

In the distribution, the RDF files appear in `[vivo]/rdf` and in `[vitro]/webapp/rdf`. These directories are merged during the build process in the usual way, with files in VIVO preferred over files in Vitro.

During the VIVO build process, the RDF files are copied to the VIVO home directory, and at runtime VIVO will read them from there.

10.4.10.5.2 The "first time"

For purposes of initialization, "first time" RDF files are loaded if the relevant data model contains no statements. Content models may also load "first time" files if the RDFService detects that its SDB-based datastore has not been initialized.

10.4.10.5.3 Initializing Configuration models

10.4.10.5.3.1 Application metadata

Function: Describes the configuration of VIVO at this site. Many of the configuration options are obsolete.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this is the first startup, read the files in rdf/applicationMetadata/firsttime.

- In Vitro, there are none
- In VIVO, `initialSiteConfig.rdf, classgroups.rdf and propertygroups.rdf`

10.4.10.5.3.2 User Accounts

Contains login credentials and assigned roles for VIVO users.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-userAccounts

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/auth/firsttime.

- In Vitro, there are none (except during Selenium testing)
- In VIVO, there are none

Every time, read the files in rdf/auth/everytime

- In Vitro, `permission_config.n3`
- In VIVO, there are none.

10.4.10.5.3.3 The Display model

This is the ABox for the display model, and contains the RDF statements that define managed pages, custom short views, and other items.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in `rdf/display/firsttime`

- In Vitro, `application.owl, menu.n3, profilePageType.n3, pageList_editableStatements.n3`
- VIVO contains its own copy of menu.n3, which overrides the one in Vitro `aboutPage.n3 menu.n3 PropertyConfig.n3 PropertyConfigSupp.n3`

Every time, read the files in rdf/display/everytime

- in Vitro, `dataGetterLabels.n3  permissions.n3 displayModelListViews.rdf  searchIndexerConfigurationVitro.n3 pageList.n3   vitroSearchProhibited.n3`
- In VIVO `homePageDataGetters.n3  vivoConceptDataGetters.n3 localeSelectionGUI.n3  vivoListViewConfig.rdf n3ModelChangePreprocessors.n3  vivoOrganizationDataGetters.n3 orcidInterfaceDataGetters.n3  vivoQrCodeDataGetter.n3 searchIndexerConfigurationVivo.n3 vivoSearchProhibited.n3`

10.4.10.5.3.4 Display TBox

The TBox for the display model.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadataTBOX

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayTbox/everytime.

- In Vitro, `displayTBOX.n3`
- In VIVO, there are none

10.4.10.5.3.5 DisplayDisplay

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata-displayModel

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayDisplay/everytime

- In Vitro, `displayDisplay.n3`
- In VIVO, there are none.

10.4.10.5.4 Initializing Content models

10.4.10.5.4.1 base ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-2

Source: named graph from the RDFService

If first setup, read the files in `rdf/abox/firsttime`

- In Vitro, there are none
- In VIVO, `geopolitical.ver1.1-11-18-11.individual-labels.rdf`

Every time, read the files in `rdf/abox/filegraph`, and create named models in the RDFService. Add them as sub-models to the base ABox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, there are none
- In VIVO `documentStatus.owl` `academicDegree.rdf` `geopolitical.abox.ver1.1-11-18-11.owl` `us-states.rdf` `continents.n3` `validation.n3` `dateTimeValuePrecision.owl` `vocabularySource.n3`
- Plus whatever data packages you may have added. See Managing Data Packages (see page 116)

10.4.10.5.4.2 base TBox

Name: http://vitro.mannlib.cornell.edu/default/asserted-tbox

Source: named graph from the RDFService (memory-mapped)

If first setup, read the files in rdf/tbox/firsttime (without subdirectories)

- In Vitro, there are none
- In VIVO, additionalHiding.n3 initialTBoxAnnotations.n3

Every time, read the files in `rdf/tbox/filegraph`, and create named models in the RDFService. Add them as sub-models to the base TBox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, `vitro-0.7.owl, vitroPublic.owl`
- In VIVO `education.owl   personTypes.n3 agent.owl   event.owl   process.owl   appControls-temp.n3 geo-political.owl publication.owl bfo-bridge.owl   grant.owl   relationship.owl bfo.owl   linkSuppression.n3 relationshipAxioms.n3 classes-additional.owl location.owl   research-resource-iao.owl   clinical.owl   object-properties.owl research-resource.owl contact-vcard.owl object-properties2.owl research.owl contact.owl   object-properties3.owl role.owl data-properties.owl objectDomains.rdf sameAs.n3 dataDomains.rdf   objectRanges.rdf service.owl dataset.owl   ontologies.owl   skos-vivo.owl date-time.owl   orcid-interface.n3 teaching.owl   dateTimeValuePrecision.owl other.owl   vitro-0.7.owl documentStatus.owl outreach.owl   vitroPublic.owl`
- Plus whatever ontology extensions you may have added

10.4.10.5.4.3 base Full

Source: a combination of base ABox and base TBox

10.4.10.5.4.4 inference ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-inf

Source: named graph from the RDFService

10.4.10.5.4.5 inference TBox

Name: http://vitro.mannlib.cornell.edu/default/inferred-tbox

Source: named graph from the RDFService (memory-mapped)

10.4.10.5.4.6 inference Full

Source: a combination of inference ABox and inference TBox

10.4.10.5.4.7 union ABox

Source: a combination of base ABox and inference ABox

10.4.10.5.4.8 union TBox

Source: a combination of base TBox and inference TBox

10.4.10.5.4.9 union Full

Source: a combination of union ABox and union TBox

10.4.11 VIVO and the Solr search engine

10.4.11.1 What is Solr?

Solr is an open-source, enterprise level search platform, available from Apache. It is based on the popular Lucene search engine. VIVO uses a standard instance of Solr, without modification. You can learn more about Solr at the Apache Solr home page[161].

VIVO maintains its data in a semantic triple-store. A triple-store is very well suited for expressing a complex, flexible web of data relationship. It is not very well suited for text-based searches. Solr provides fast searching with features like

- weighted results by field,
- searching by the stems of words, rather than exact matches,
- faceted search results,
- and much more.

Solr provides these features much more efficiently than a triple-store would.

Solr maintains its own index of data, which reflects the contents of the triple-store. As the data in VIVO changes, the contents of the Solr index must change also. In most cases this happens automatically, but not always. Sometimes the search index must be rebuilt to bring it into synchronization with the triple-store. See the section below called "When is the index updated?[162]" for more information.

Solr is implemented as a self-contained web application, separate from VIVO. At most VIVO sites, Solr and VIVO run on the same machine, but this is not the only possible configuration. It is possible to host Solr on a different computer from VIVO.

---

[161] https://lucene.apache.org/solr/

[162] https://wiki.duraspace.org/display/VIVODOC111x/VIVO+and+the+Solr+search+engine#VIVOandtheSolrsearchengine-Whenistheindexupdated?

In a typical VIVO installation, Solr is hidden behind VIVO, and the users do not access it directly. In general, they don't know that Solr exists as an application. Before releasing VIVO to the public, a site admin should take steps to secure their Solr installation to ensure only administrators and the VIVO application can modify the search index.

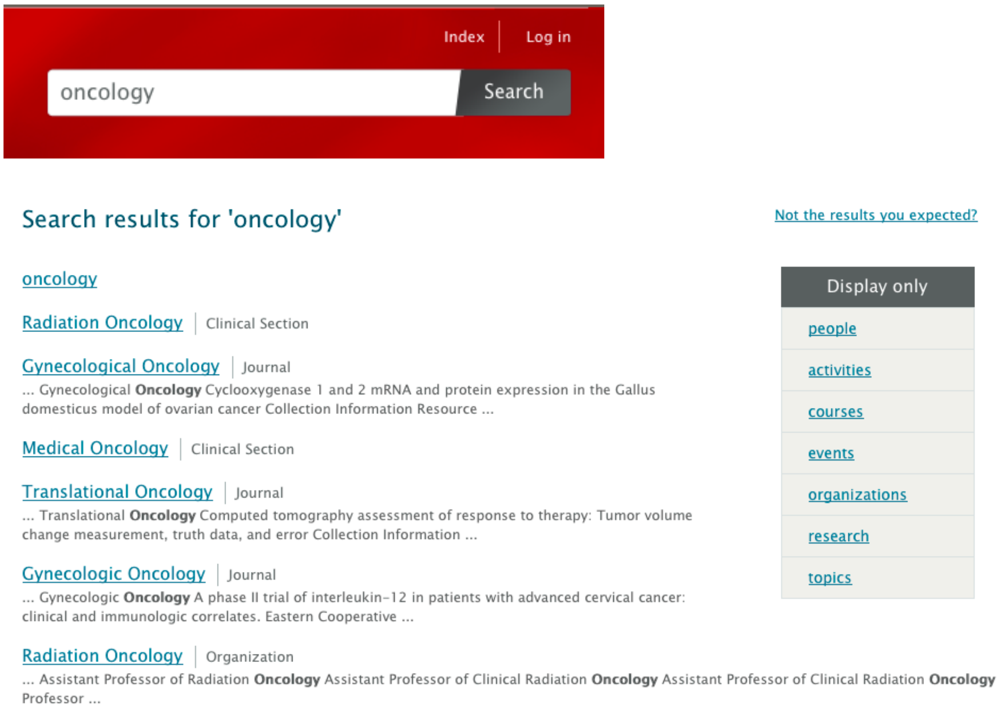
10.4.11.2 How does VIVO use Solr?

VIVO uses the Solr search engine in two ways:

- as a service to the end user,
- as a tool within the structure of the application.

10.4.11.2.1 Solr for the end user

Like many web sites, VIVO includes a search box on every page. The person using VIVO can type a search term, and see the results. This search is conducted by Solr, and the results are formatted and displayed by VIVO.



Search results for 'oncology'

oncology

Radiation Oncology | Clinical Section

Gynecological Oncology | Journal
... Gynecological **Oncology** Cyclooxygenase 1 and 2 mRNA and protein expression in the Gallus domesticus model of ovarian cancer Collection Information Resource ...

Medical Oncology | Clinical Section

Translational Oncology | Journal
... Translational **Oncology** Computed tomography assessment of response to therapy: Tumor volume change measurement, truth data, and error Collection Information ...

Gynecologic Oncology | Journal
... Gynecologic **Oncology** A phase II trial of interleukin-12 in patients with advanced cervical cancer: clinical and immunologic correlates. Eastern Cooperative ...

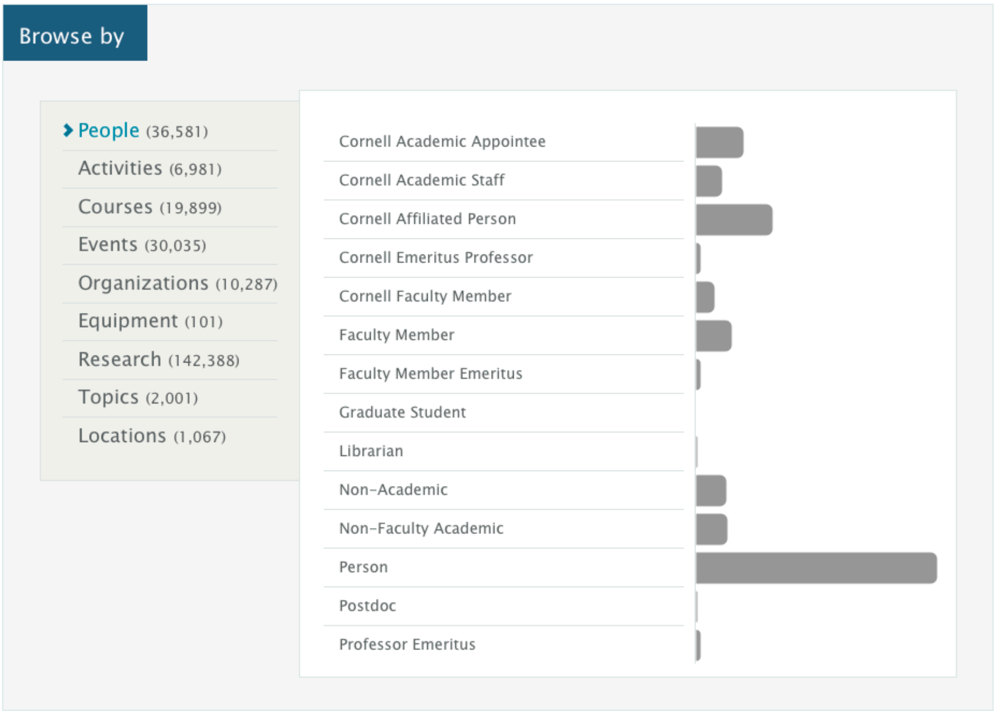
Radiation Oncology | Organization
... Assistant Professor of Radiation **Oncology** Assistant Professor of Clinical Radiation **Oncology** Assistant Professor of Clinical Radiation **Oncology** Professor ...

Solr allows for a "faceted" search, and VIVO displays the facets on the right side of the results page. These allow the user to filter the search results, showing only entries for people, or for organizations, etc.

10.4.11.2.2 Solr within VIVO

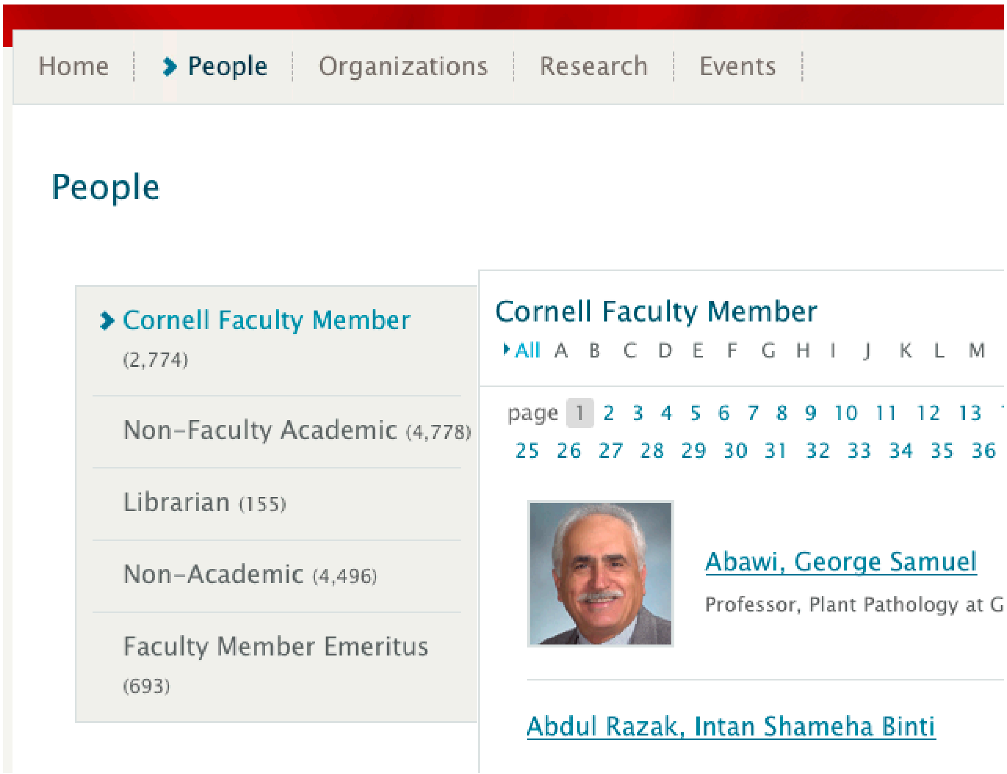
VIVO is based around an RDF triple-store, which holds all of its data. However, there are some tasks that a search engine can do much more quickly than a triple-store. Some of the fields in the Solr search index were put there specifically to help with these tasks.

For example, the browse area on the home page shows how many individuals VIVO holds for each class group.


Browse by
People (36,581)
Activities (6,981)
Courses (19,899)
Events (30,035)
Organizations (10,287)
Equipment (101)
Research (142,388)
Topics (2,001)
Locations (1,067)
Cornell Academic Appointee
Cornell Academic Staff
Cornell Affiliated Person
Cornell Emeritus Professor
Cornell Faculty Member
Faculty Member
Faculty Member Emeritus
Graduate Student
Librarian
Non-Academic
Non-Faculty Academic
Person
Postdoc
Professor Emeritus

VIVO could produce this data by issuing a SPARQL query against its data model. However, this would take several seconds for a large site, and we do not want the user to wait that long to see the home page. To avoid this delay, the class group of each individual is stored in the Solr record for that individual. Solr can count these fields very quickly, so VIVO issues a Solr query against the index, and displays the results on the home page.

Record counts on VIVO's index pages are obtained using the same type of Solr query.


Home
People
Organizations
Research
Events
People
Cornell Faculty Member (2,774)
Non-Faculty Academic (4,778)
Librarian (155)
Non-Academic (4,496)
Faculty Member Emeritus (693)
Cornell Faculty Member
All A B C D E F G H I J K L M
page 1 2 3 4 5 6 7 8 9 10 11 12 13
25 26 27 28 29 30 31 32 33 34 35 36
Abawi, George Samuel
Professor, Plant Pathology at G
Abdul Razak, Intan Shameha Binti

10.4.11.3 How is Solr created and configured?

Solr is an external component to VIVO and must be installed separately. See "Configure and Start Solr[163]" for detailed installation instructions.

The behavior of Solr depends extensively on its configuration files. These are stored in a directory that is called the Solr Home directory. When VIVO runs, the Solr search index is built inside the Solr Home directory.

10.4.11.4 The search index

10.4.11.4.1 What is in the index?

The Solr search index contains one record for each Individual in VIVO, unless that individual is explicitly excluded from the index. Exclusions are usually made for individuals that represent "context nodes" in the VIVO data model.

For example, if a professor teaches a course, the search index will contain:

- a record for the professor
- a record for the course

The VIVO data model also contains an individual that represents this teaching activity. That individual will be excluded from the index, since users would almost certainly prefer to find the teacher or the course in their search results, rather than the concept that connects the two.

---

[163] https://wiki.duraspace.org/display/VIVODOC111x/Installing+VIVO#InstallingVIVO-ConfigureandStartSolr

10.4.11.4.2 What is in each record?

Each record in the search index contains several fields (see the chart below). The most commonly used field is `alltext`, In the record for a faculty member, `alltext` will contain her name, the name of her department, the names of her classes, the names of her papers and grants, etc. So, if you search for "Carpenter", you might see results for people named Carpenter, people in the Carpentry department, people who have written papers about carpentry, or have worked on grants about carpentry. You would also see results for the department itself, for the papers, and for the grants.

Solr index fields, VIVO 1.6

DocId	nameRaw	PREFERRED_TITLE
URI	nameText	siteURL
ALLTEXT	nameLowercase	siteName
ALLTEXTUNSTEMMED	nameLowercaseSingleValued	THUMBNAIL
classgroup	nameUnstemmed	THUMBNAIL_URL
type	nameStemmed	indexedTime
mostSpecificTypeURIs	acNameUntokenized	timestamp
BETA	acNameStemmed	etag
PROHIBITED_FROM_TEXT_RESULTS	NAME_PHONETIC	

10.4.11.4.3 When is the index updated?

10.4.11.4.3.1 During normal operation

When an individual is added, edited, or deleted through VIVO's user interface, Solr is given the new information and the index is updated.

VIVO administrators may also make changes to the data using the Advanced Data Tools, which are accessible from the Site Administration page. These tools also pass the data changes to Solr, so the index is kept current with the data.

Finally, data can be modified using the The SPARQL Update API (see page 372). Again, Solr receives the changes and the index remains current.

10.4.11.4.3.2 On demand

Some tools, such as the VIVO Harvester, bypass VIVO and write directly to the data store. Solr is not notified when these tools are used, and the data becomes out of sync with the search index.

Other circumstances can cause issues with the search index. Perhaps a problem required you to restore your database to a backup, but you did not restore your search index at the same time. Perhaps you are developing a modification for VIVO, and you have emptied your database in order to test it. Perhaps VIVO crashed while data was being ingested.

In any of these circumstances, the solution is to login to VIVO as an administrator, navigate to the `Site Administration` page and click on `Rebuild search index`.

The existing search index remains in place while the new index is being built. When the rebuild is complete, the new index replaces the old one, and the old index is deleted.

10.4.11.4.4 Customizing the index

10.4.11.4.4.1 Creating custom fields

There are two parts to adding a custom field to VIVO's search index, defining the VIVO SPARQL query and defining the search engine's fields that will be populated.

Custom queries can be added to any file in the vivo-home/rdf/display/everytime directory (or any other file directory read by VIVO during startup). Using searchIndexerConfigurationVivo.n3[164] as a template, create a query that returns the data you wish to add to the search index. For example, if you wanted to create a search field for tracking open access publications you flag the content with a custom data property:

```
@prefix : <http://vitro.mannlib.cornell.edu/ns/vitro/ApplicationSetup#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

:vivodocumentModifier_openAccess
    a
<java:edu.cornell.mannlib.vitro.webapp.searchindex.documentBuilding.SelectQueryDocume
ntModifier> ,

<java:edu.cornell.mannlib.vitro.webapp.searchindex.documentBuilding.DocumentModifier>
 ;
    rdfs:label "open access" ;
    :hasTargetField "open_access_s" ;
    :hasSelectQuery """
```

---

[164] https://github.com/vivo-project/VIVO/blob/develop/home/src/main/resources/rdf/display/everytime/searchIndexerConfigurationVivo.n3

```
        PREFIX wos: <http://webofscience.com/ontology/wos#>
        SELECT ?status
        WHERE {
            ?uri wos:openAccess ?status .
        }
        LIMIT 1
        """ .
```

Second (at least in the case of Solr), you must add the new field to the search index schema. For Solr 6 and older, this can be defined directly in Solr's schema.xml at any time (schema.xml included with Vitro prior to v1.11 for reference[165]). For later versions of Solr, schema.xml will be read during core creation, but will not read changes after the fact. Instead, you may add new fields using Solr's Schema API. For the above example, you could post:

```
curl -X POST -H 'Content-type:application/json' --data-binary '{
  "add-field": {
      "name":"open_access_s",
      "type":"text",
      "multiValued":false,
      "stored":true}
  }' http://localhost:8983/api/cores/vivocore/schema
```

10.4.11.4.4.2 Excluding specific classes from the search index

Exclusions can be added to any file in the vivo-home/rdf/display/everytime directory (or any other file directory read by VIVO during startup). You may want to overwrite searchIndexerConfigurationVivo.n3[166] to keep the search configuration in one place. You can exclude individual class types:

```
@prefix : <http://vitro.mannlib.cornell.edu/ns/vitro/ApplicationSetup#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:vivoSearchExcluder_typeExcluder
    a
<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.ExcludeBasedOnType> ,

<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.SearchIndexExcluder> ;
    :excludes
        "http://purl.org/NET/c4dm/event.owl#Event" .
```

... or all class types within a certain namespace:

```
@prefix : <http://vitro.mannlib.cornell.edu/ns/vitro/ApplicationSetup#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

---

[165] https://github.com/vivo-project/Vitro/blob/vitro-1.10.0/home/src/main/resources/solr/conf/schema.xml

[166] https://github.com/vivo-project/VIVO/blob/develop/home/src/main/resources/rdf/display/everytime/searchIndexerConfigurationVivo.n3

```
:vivoSearchExcluder_namespaceTypeExcluder
    a
<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.ExcludeBasedOnTypeNames
pace> ,

<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.SearchIndexExcluder> ;
    :excludes
        "http://purl.org/NET/c4dm/event.owl#" .
```

... or the namespace of the object URI:

```
@prefix : <http://vitro.mannlib.cornell.edu/ns/vitro/ApplicationSetup#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:vivoSearchExcluder_namespaceExcluder
    a
<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.ExcludeBasedOnNamespace
> ,

<java:edu.cornell.mannlib.vitro.webapp.searchindex.exclusions.SearchIndexExcluder> ;
    :excludes
        "http://localhost/private/" .
```

10.4.11.5 How does VIVO contact Solr?

VIVO must be configured to communicate with Solr. This is accomplished by updating the "runtime.properties[167]" to point to the URL of Solr. More installation details can be found in the "Configure and Start Solr[168]" documentation.

10.4.12 Image storage

The uploaded image files are identified by a combination of URI and filename. The URI is used as the principal identifier so we don't need to worry about collisions if two people each upload an image named "image.jpg". The filename is retained so the user can use their browser to download their image from the system and it will be named as they expect it to be.

We wanted a way to store thousands of image files so they would not all be in the same directory. We took our inspiration from the PairTree[169] folks, and modified their algorithm to suit our needs. The general idea is to store files in a multi-layer directory structure based on the URI assigned to the file.

Let's consider a file with this information:

URI	http://vivo.mydomain.edu/individual/n3156

---

[167] https://github.com/vivo-project/VIVO/blob/develop/home/src/main/resources/config/example.runtime.properties#L85
[168] https://wiki.duraspace.org/display/VIVODOC111x/Installing+VIVO#InstallingVIVO-ConfigureandStartSolr
[169] https://wiki.ucop.edu/display/Curation/PairTree

Filename	`lily1.jpg`

In this example, we assume that VIVO's home directory is at `/usr/local/vivo`.

We want to turn the URI into the directory path, but the URI contains prohibited characters. Using a PairTree-like character substitution, we might store it at this path:

```
/usr/local/vivo/uploads/file_storage_root/http+==vivo.mydomain.edu=individual=n3156/lily1.jpg
```

Using that scheme would mean that each file sits in its own directory under the storage root. At a large institution, there might be hundreds of thousands of directories under that root.

By breaking this into PairTree-like groupings, we insure that all files don't go into the same directory. Limiting to 3-character names will insure a maximum of about 30,000 files per directory. In practice, the number will be considerably smaller. So then it would look like this:

```
/usr/local/vivo/uploads/file_storage_root/htt/p+=/=vi/vo./myd/oma/in./edu/=in/div/idu/al=/n31/56/
lily1.jpg
```

But almost all of our URIs will start with the same namespace, so the namespace just adds unnecessary and unhelpful depth to the directory tree. We assign a single-character prefix to that namespace, using the `file_storage_namespaces.properties` file in the uploads directory, like this:

```
a = http://vivo.mydomain.edu/individual/
```

And our URI now looks like this:

```
a~n3156
```

Which translates to:

```
/usr/local/vivo/uploads/file_storage_root/a~n/315/6/lily1.jpg
```

So what we hope we have implemented is a system where:

- Files are stored by URI and filename.
- File paths are constructed to limit the maximum number of files in a directory.
- "Illegal" characters in URIs or filenames will not cause problems.
  - even if a character is legal on the client and illegal on the server.

- Frequently-used namespaces on the URIs can be collapsed to short prefix sequences.
- URIs with unrecognized namespaces will not cause problems.

By the way, almost all of this is implemented in `edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelper` and illustrated in `edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelperTest`

10.4.12.1 Access images after changing the default namespace

If you are moving images from one server to another, with no change in the URL, it should be sufficient to just move the VIVO home directory with no changes. VIVO will find `file_storage_namespaces.properties` and `file_storage_root` in `[home]/uploads`, and everything still works.

If you are changing to a new URL, I presume that you are changing to a new default namespace. Have you used the “Change Namespace of Resources" tool? (http://localhost:8082/vivo/ingest?action=renameResource)

So your file individual has changed from the old URI `http://localhost:8082/vivo/individual/n187` to the new URI `http://logics.emap.fgv.br:8080/vivo/individual/n187`

However, `file_storage_namespaces.properties` does not know how to translate this new namespace.

One way to cope with this is to edit `file_storage_namespace.properties` accordingly, adding this line: `b = http://logics.emap.fgv.br:8080/vivo/individual/`

and rename your `[home]/uploads/file_storage_root/a~n` directory to `[home]/uploads/file_storage_root/b~n`

The new URI now translates to `b~n187`, and the file which is now stored at `[home]/uploads/file_storage_root/b~n/187/servletrecuperafoto.jpeg` is accessible by its new URI.

10.4.12.2 How are Images represented in the Model?

When an image file is uploaded via the GUI, the process is something along these lines:

- upload the image file, and store in a temporary location.
- ask the user for a cropping square to be used in producing the thumbnail.

- create a URI for the image file surrogate object, and a URI for the image file bytestream object.
- create a URI for the thumbnail surrogate object, and a URI for the thumbnail bytestream object.
- hand the image file bytestream URI and the temporary file to the File Storage system, which will create a permanent storage.
- generate a 115 by 115 JPEG thumbnail image from the main image and the cropping square.
- hand the thumbnail image stream and the thumbnail bytestream URI to the File Storage system, which will create a permanent storage.
- create a thumbnail bytestream object in the model.
- create a thumbnail surrogate object in the model, storing the filename of the thumbnail, the mime type of the thumbnail, and the URI of the thumbnail bytestream.
- create a main image bytestream object in the model.
- create a main image surrogate object in the model, storing the filename of the main image, the mime type of the main image, and the URI of the main image bytestream.
- link the main image surrogate object to the person object.

These are no more than a multitude of technical details, except: how do you find an appropriate region of the image to use as the thumbnail?

Generating the thumbnail itself can be quite problematic if the initial image is a GIF or PNG with transparency.

For an individual on my test installation (in N3, if I remember how to write it)

10.4.12.2.1 INDIVIDUAL

```
<http://vivo.mydomain.edu/individual/n1451>
    http://vitro.mannlib.cornell.edu/ns/vitro/public#mainImage
            http://vivo.mydomain.edu/individual/n1674.
```

10.4.12.2.2 IMAGE FILE SURROGATE

```
<http://vivo.mydomain.edu/individual/n1674>
    http://vitro.mannlib.cornell.edu/ns/vitro/public#thumbnailImage
            http://vivo.mydomain.edu/individual/n5863;
    http://vitro.mannlib.cornell.edu/ns/vitro/public#downloadLocation
            http://vivo.mydomain.edu/individual/n3156;
    http://vitro.mannlib.cornell.edu/ns/vitro/public#mimeType
            "image/jpeg";
    http://vitro.mannlib.cornell.edu/ns/vitro/public#filename
```

```
            "lily1.jpg";
        http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
                "2010-10-18T09:51:58";
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://vitro.mannlib.cornell.edu/ns/vitro/public#File;
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://www.w3.org/2002/07/owl#Thing.
```

10.4.12.2.3 IMAGE FILE BYTESTREAM

```
    <http://vivo.mydomain.edu/individual/n3156>
        http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
                "2010-10-18T09:51:57";
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://vitro.mannlib.cornell.edu/ns/vitro/
public#FileByteStream;
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://www.w3.org/2002/07/owl#Thing.
```

10.4.12.2.4 THUMBNAIL SURROGATE

```
    <http://vivo.mydomain.edu/individual/n5863>
        http://vitro.mannlib.cornell.edu/ns/vitro/public#downloadLocation
                http://vivo.mydomain.edu/individual/n5889;
        http://vitro.mannlib.cornell.edu/ns/vitro/public#mimeType
                "image/jpeg";
        http://vitro.mannlib.cornell.edu/ns/vitro/public#filename
                "thumbnail_lily1.jpg";
        http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
                "2010-10-18T09:52:12";
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://vitro.mannlib.cornell.edu/ns/vitro/public#File;
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

```
                http://www.w3.org/2002/07/owl#Thing.
```

10.4.12.2.5 THUMBNAIL BYTESTREAM

```
    <http://vivo.mydomain.edu/individual/n5889>
        http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
                "2010-10-18T09:52:12";
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://vitro.mannlib.cornell.edu/ns/vitro/
public#FileByteStream;
        http://www.w3.org/1999/02/22-rdf-syntax-ns#type
                http://www.w3.org/2002/07/owl#Thing;
```

The file system looks something like this:

File storage properties file: /usr/local/vivo/uploads/file_storage_namespace.properties

```
a = http://vivo.mydomain.edu/individual/
```

Main image:

```
/usr/local/vivo/uploads/file_storage_root/a~n/315/6/lily1.jpg
```

Thumbnail:

```
/usr/local/vivo/uploads/file_storage_root/a~n/588/9/thumbnail_lily1.jpg
```

Note: The file storage system does "laundering" on the filenames, in order to allow files with special characters to be stored in a portable manner (e.g., Linux or Windows).

Jim

10.4.12.2.6 A summary from Eliza Chan

Excerpted from a message[170] in the vivo-dev-all archive, by Eliza Chan, dated 2010-11-04 16:08

As an experiment tested on localhost, when the pictures were uploaded using a "non-traditional" method, i.e. copying directly to the folder /usr/local/vivo/data/uploads/file_storage_root/a~n, the content under primary tab became blank (see attachment localhost_vivo_mainTab.tiff). Pictures did show up but only when the primary tab content was clicked (see attachment localhost_vivo_tabContent.tiff). The reason for copying directly to the folder was to save the work for doing manual upload of about 1000 photos.

The way it was done was as follows:

1. Create RDF for images and add to the VIVO site, e.g.

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/cwid-gwa2001">
<j.2:mainImage rdf:resource="http://localhost:8080/vivo/individual/mainImage-gwa2001"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
<rdf:type rdf:resource="http://vivoweb.org/ontology/core#FacultyMember"/>
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/mainImage-gwa2001">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.2:downloadLocation rdf:resource="http://localhost:8080/vivo/individual/n1229119954939"/>
<j.2:thumbnailImage rdf:resource="http://localhost:8080/vivo/individual/thumbnailImage-gwa2001"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<j.2:mimeType xml:lang="en">image/jpeg</j.2:mimeType>
<j.2:filename xml:lang="en">_main_image_gwa2001.jpg</j.2:filename>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#File"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/n1229119954939">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream"/>
</rdf:Description>
```

---

[170] https://sourceforge.net/mailarchive/message.php?msg_id=26544669

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/thumbnailImage-gwa2001">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.2:downloadLocation rdf:resource="http://localhost:8080/vivo/individual/n12291199549391"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<j.2:mimeType xml:lang="en">image/jpeg</j.2:mimeType>
<j.2:filename xml:lang="en">gwa2001.jpg</j.2:filename>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#File"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/n12291199549391">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream"/>
</rdf:Description>
```

2. Copy images to the following folders:

```
/usr/local/vivo/data/uploads/file_storage_root/a~n/122/911/995/493/9/_main_image_gwa2001.jpg
/usr/local/vivo/data/uploads/file_storage_root/a~n/122/911/995/493/91/gwa2001.jpg
```

10.4.12.2.7 Update on "alias URL" and "directDownloadUrl" property

You can retrieve an image file by asking for the Individual page of its FileByteStream. For example,

```
http://localhost:8080/vivo/individual/n4898
```

VIVO will see that this particular individual is a FileByteStream, and will redirect your browser to the "alias URL" for that image. In this case:

```
http://localhost:8080/vivo/file/n4898/john_doe.jpg
```

This redirection means that the image shown in your browser has a name that you will recognize, with an appropriate file type. If you choose "Save Image" in your browser, the default filename will be suitable for the image.

However, this redirection implies additional overhead. Pages local to VIVO calculated the alias URL and used it as the "src" property on the image, avoiding the redirection. But because the "alias URL" was not present in the RDF, it was not available to external applications, which resulted in excessive load times for pages that displayed dozens of images.

The directDownloadUrl" property of FileByteStream objects contains the "alias URL", is created when the image is ingested, and is used both by VIVO and by external applications when displaying images.

Accordingly, the FileByteStream examples shown above must now look like this instead:

10.4.12.2.7.1 IMAGE FILE BYTESTREAM

```
  <http://vivo.mydomain.edu/individual/n3156>
      http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
              "2010-10-18T09:51:57";
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              http://vitro.mannlib.cornell.edu/ns/vitro/
public#FileByteStream;
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              http://www.w3.org/2002/07/owl#Thing.
      http://vitro.mannlib.cornell.edu/ns/vitro/public#directDownloadUrl
              "/file/n3156/lily1.jpg"
```

10.4.12.2.7.2 THUMBNAIL BYTESTREAM

```
  <http://vivo.mydomain.edu/individual/n5889>
      http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
              "2010-10-18T09:52:12";
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              http://vitro.mannlib.cornell.edu/ns/vitro/
public#FileByteStream;
      http://www.w3.org/1999/02/22-rdf-syntax-ns#type
              http://www.w3.org/2002/07/owl#Thing;
      http://vitro.mannlib.cornell.edu/ns/vitro/public#directDownloadUrl
```

```
            "/file/n5889/thumbnail_lily1.jpg"
```

10.5 Configuration Reference

10.5.1 Overview

VIVO's operation can be determined by setting corresponding properties in `runtime.properties`.

10.5.2 VIVO Runtime Properties

Property	**Description**
`Vitro.defaultNamespace = http://vivo.mydomain.edu/individual/`	This namespace will be used when generating URIs for objects created in the editor. In order to serve linked data, the default namespace must be composed as follows (optional elements in parentheses): scheme + server_name (+ port) (+ servlet_context) + "/individual/" For example, Cornell's default namespace is: http://vivo.cornell.edu/individual/
`rootUser.emailAddress = vivo_root@mydomain.edu`[171]	The email address of the root user for the VIVO application. The password for this user is initially set to "rootPassword", but you will be asked to change the password the first time you log in.
`argon2.parallelism =1`	For argon2i password handling. A parallelism degree defines the number of parallel threads
`argon2.memory = 1024`	For argon2i password handling. A memory cost defines the memory usage, given in kilobytes

---

[171] mailto:vivo_root@mydomain.edu

Property	Description
`argon2.time = 1000`	For argon2i password handling. A time cost defines the amount of computation realized and therefore the execution time, given in a number of iterations. For determining the optimal values of the parameters for your setup please refer to the white paper section 9 https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf

`VitroConnection.DataSource.url = jdbc:mysql://localhost/vitrodb (see page 434)`
`VitroConnection.DataSource.username = vitrodbUsername`
`VitroConnection.DataSource.password = vitrodbPassword`
	The basic parameters for a database connection. Change the end of the URL to reflect your database name (if it is not "vitrodb"). Change the username and password to match the authorized database user you created.

`email.smtpHost = smtp.mydomain.edu`[172]
`email.replyTo = vivoAdmin@mydomain.edu`[173]
 	
`# email.port = 25`
`# email.username = username`
`# email.password = password`
	Email parameters which VIVO can use to send mail. If these are left empty, the "Contact Us" form will be disabled and users will not be notified of changes to their accounts.

`vitro.local.solr.url = http://localhost:8080/vivosolr`	URL of Solr context used in local VIVO search. This will usually consist of: scheme + server_name + port + vivo_webapp_name + "solr" In the standard installation, the Solr context will be on the same server as VIVO, and in the same Tomcat instance. The path will be the VIVO webapp.name[174] (specified in build.properties) + "solr" Example: vitro.local.solr.url = http://localhost:8080/vivosolr
`selfEditing.idMatchingProperty = http://vivo.mydomain.edu/ns#networkId`	Associates user accounts with user profiles. URI of a datatype property. If the value of the property is the same on the user's profile and the user's account (specified via "Matching ID"), the profile is associated with the account.

---

[172] http://smtp.mydomain.edu
[173] mailto:vivoAdmin@mydomain.edu
[174] http://webapp.name

Property	Description
`externalAuth.netIdHeaderName = remote_userID`	If an external authentication system such as Shibboleth or CUWebAuth is to be used, this property says which HTTP header will contain the user ID from the authentication system. If such a system is not to be used, leave this commented out. See Using an external authentication system[175]
`VitroConnection.DataSource.pool.maxActive = 40`	The maximum number of active connections in the database connection pool. Increase this value to support a greater number of concurrent page requests.
`VitroConnection.DataSource.pool.maxIdle = 10`	The maximum number of database connections that will be allowed to remain idle in the connection pool. Default is 25% of the maximum number of active connections.

`VitroConnection.DataSource.dbtype = MySQL`
`VitroConnection.DataSource.driver = com.mysql.jdbc.Driver`
`VitroConnection.DataSource.validationQuery = SELECT 1`
	Parameters to change in order to use VIVO with a database other than MySQL. These parameters allow you to change the relational database that is used as the back end for Jena SDB. If you want to use a triple store other than SDB, you will need to edit applicationSetup.n3. See the installation instructions for more details.

`OpenSocial.shindigURL = http://localhost:8080/shindigorng`	For OpenSocial integration, the base URL of the ORNG Shindig server. Usually, this is the same host and port number as VIVO itself, with a context path of "shindigorng".
`OpenSocial.tokenService = myhost.mydomain.edu`[176]`:8777`	For OpenSocial integration, The host name and port number of the service that provides security tokens for VIVO and Shindig to share. For now, the host name must be the actual host, not "localhost" or "127.0.0.1" The port number must be 8777
`OpenSocial.tokenKeyFile = /usr/local/vivo/data/shindig/openssl/securitytokenkey.txt`	For OpenSocial integration. The path to the key file that will be used when generating security tokens for VIVO and shindig to share.
`OpenSocial.sandbox = True`	For OpenSocial integration. Only set sandbox to True for dev/test environments. Comment out or set to False in production

---

[175] https://wiki.lyrasis.org/display/VTDA/Using+an+external+authentication+system
[176] http://myhost.mydomain.edu

Property	**Description**
`RDFService.languageFilter = false`	Show only the most appropriate data values based on the Accept-Language header supplied by the browser. Default is false if not set.
`languages.forceLocale = en_US`	Force VIVO to use a specific language or Locale instead of those specified by the browser. This affects RDF data retrieved from the model, if RDFService.languageFilter is true. This also affects the text of pages that have been modified to support multiple languages.
`languages.selectableLocales = en_US, es_GO`	A list of supported languages or Locales that the user may choose to use instead of the one specified by the browser. Selection images must be available in the i18n/images directory of the theme. This affects RDF data retrieved from the model, if RDFService.languageFilter is true. This also affects the text of pages that have been modified to support multiple languages. This should not be used with languages.forceLocale, which will override it.

`orcid.clientId = 0000-0000-0000-000X`
`orcid.clientPassword = 00000000-0000-0000-0000-000000000000`
`orcid.webappBaseUrl = http://localhost:8080/vivo`
`orcid.externalIdCommonName = VIVO Cornell Identifier`
`orcid.apiVersion = 2.0`
`orcid.api = release | sandbox`
	ORCiD integration parameters. See Activating the ORCID integration (see page 311)

`google.maps.key=`	To use the Google Maps (e.g. Map of Science), you need to have a key for Google Maps. See https://developers.google.com/maps/documentation/javascript/get-api-key When you have a key, uncomment the line below and add it here
`resource.altmetric=disabled`	Uncomment and set this to disabled if you don't want AltMetric badges
`resource.altmetric.displayto=right`	Display the badge to the left or right of the title (default = right). Options: left, right

Property	Description
`resource.altmetric.badge-type=donut`	Badge type to display (default = donut) Options: See AltMetric documentation[177] - recommended settings: donut, medium-donut
`resource.altmetric.hide-no-mentions=true`	Hide the badge if there are no mentions (default = true) Options: true, false
`resource.altmetric.badge-popover=right`	Display more details about the score when you hover over the badge (default = right) Options, right, left, up, down
`resource.altmetric.badge-details=right`	Display extended details alongside the badge (default = none)
`homePage.geoFocusMaps=enabled`	When the following flag is set to enabled, the VIVO home page displays a global map highlighting the geographical focus of foaf:person individuals. See Home page customizations (see page 154)
`multiViews.profilePageTypes=enabled`	VIVO supports the simultaneous use of a full foaf:Person profile page view and a "quick" page view that emphasizes the individual's webpage presence. Implementing this feature requires an installation to develop a web service that captures images of web pages or to use an existing service outside of VIVO. See Multiple profile types for foaf:Person (see page 240)
`http.createCacheHeaders = true`	Tell VIVO to generate HTTP headers on its responses to facilitate caching the profile pages that it creates. See Use HTTP caching to improve performance (see page 322) Developers will likely want to leave caching disabled, since a change to a Freemarker template or to a Java class would not cause the page to be considered stale.
`harvester.location = /usr/local/vivo/harvester/`	Absolute path on the server of the Harvester root directory. You must include the final slash. Setting a value for harvester.location indicates that the Harvester is installed at this path. This will enable the Harvester functions in the Ingest Tools page.

---

[177] https://api.altmetric.com/embeds.html#badge-types

Property	Description
`visualization.topLevelOrg = http://vivo.mydomain.edu/individual/topLevelOrgURI`	The temporal graph visualization is used to compare different organizations/people within an organization on parameters like number of publications or grants. By default, the app will attempt to make its best guess at the top level organization in your instance. If you're unhappy with this selection, uncomment out the property below and set it to the URI of the organization individual you want to identify as the top level organization. It will be used as the default whenever the temporal graph visualization is rendered without being passed an explicit org. For example, to use "Ponce School of Medicine" as the top organization: visualization.topLevelOrg = http://vivo.psm.edu/individual/n2862
`visualization.temporal = enabled`	The temporal graph visualization can require extensive machine resources. This can have a particularly noticeable impact on memory usage if The organization tree is deep, The number of grants and publications is large. VIVO 1.3 release mitigates this problem by the way of a caching mechanism hence we can safely set this to be enabled by default.
`proxy.eligibleTypeList = http://xmlns.com/foaf/0.1/Person, http://xmlns.com/foaf/0.1/Organization`	Types of individual for which we can create proxy editors. If this is omitted, defaults to http://www.w3.org/2002/07/owl#Thing

Property	Description
`Vitro.reconcile.defaultTypeList = http://vivoweb.org/ontology/core#Role, core:Role; http://vivoweb.org/ontology/core#AcademicDegree, core:Academic Degree; http://purl.org/NET/c4dm/event.owl#Event, event:Event; http://vivoweb.org/ontology/core#Location, core:Location; http://xmlns.com/foaf/0.1/Organization, foaf:Organization; http://xmlns.com/foaf/0.1/Person, foaf:Person; http://purl.obolibrary.org/obo/IAO_0000030, obo:IAO_0000030`	Default type(s) for Google Refine Reconciliation Service. The format for this property is id, name; id1, name1; id2, name2 etc. For more information, see Service Metadata from this page: https://github.com/OpenRefine/OpenRefine/wiki/Reconciliation-Service-Api

10.6 Directories and Files

10.6.1 Overview

The directory structure below is for the VIVO source distribution. The binary distribution omits some directories. These are noted below.

The Vitro source distribution has an analogous structure.

10.6.2 High Level Directories

Directory	Description
`./api`	Java source for the webapp
`./home`	RDF and other files needed to load the webapp

Directory	Description
`./installer`	Files used by the Maven installer
`./paper`	The "'VIVO: a system for research discovery' tags" paper
`./selenium`	VIVO Selenium Tests. See http://docs.seleniumhq.org
`./webapp`	Templates and other files for building the webapp

10.6.3 Directory Structure

Directory	Description
`./api/src/main`	VIVO source files. Will not be present in the binary distribution
`./api/src/test`	VIVO source test files (java files and resources). Will not be present in the binary distribution
`./api/target/surefire-reports`	JUnit test reports
`./api/target/test-classes`	Compiled `./api/src/test` classes and resources
`./home/rdf/abox`	Data about specific named “individuals” and their relationships (the “assertion box” or ABox)
`./home/rdf/applicationMetadata`	Data for configuring the VIVO initial site information, class and property groups
`./home/rdf/auth`	Data for configuring permissions for the VIVO application

Directory	Description

`./home/rdf/display,`
`./home/rdf/displayDisplay,`
`./home/rdf/displayTbox`
	Data for configuring the VIVO application display options - menu, profile page, etc.

`./home/rdf/tbox`	Data about the structure of ontologies (the “terminological box” or TBox)
`./home/upgrade/knowledgeBase`	Files used for migration of data between different versions of VIVO
`./home/uploads/file_storage_root`	Images and other assets preserved in the VIVO application
`./installer/home`	Resources for initial building of VIVO_HOME directory by using ./home resources
`./installer/webapp`	Resources for initial building of web application by using ./webapp resources
`./paper`	The paper entitled "'VIVO: a system for research discovery' tags" written in the Markdown markup language.
`./selenium/src`	The selenium Java source files
`./selenium/test-output`	The selenium test reports
`./webapp/src`	Source files for building VIVO UI
`./webapp/target`	The compiled and built VIVO UI

10.7 Freemarker Template Variables and Directives

Template variables are made available to render dynamic content within the application. To print a variable's value in FreeMarker, use the following syntax:
${variableName}

Some variables have methods which can be used to return a value or perform a task such as adding a stylesheet or script to the <head> element.
${stylesheets.add('<link rel="stylesheet" href="mystylesheet.css" />')}
${headScripts.add(<script type="text/javascript" src="myscript.js"></script>)}

Special template directives provide debugging features that assist in template development.
<@describe var="stylesheets" />
(describe the methods callable on a template variable)

<@dump var ="stylesheets" />
(dump the contents of a template variable)

<@dumpAll />
(dump the contents of the template data model)

A sample page at *

`http://yourLocalInstance.com/freemarkersamples`

* demonstrates most of the methods and directives available within a template. The template file responsible for this page is vitro-core/webapp/web/templates/freemarker/body/samples.ftl.

10.8 Graph Reference

10.8.1 Overview

VIVO stores its information in graphs – named collections of triples. Graphs keep data organized by kind, and provide the opportunity for different access rights and management practices to be applied at the graph level. All graphs are available to the VIVO SPARQL query interface. When using SPARQL to query the VIVO data, one does not need to know the graph the data is contained in. Triples in all graphs are available to the query. When updating data in VIVO using CONSTRUCT or UPDATE, knowledge of the graph may be necessary.

Here we show how to list the graphs in a VIVO, and provide a reference for the purpose of each graph.

10.8.2 Listing the graphs used by VIVO

To list the graphs being used by your VIVO, you can run the SPARQL query shown below. Caution: If you have a significant amount of data in your VIVO, the query may take quite a while to run. With tens of thousands of entities in your VIVO, the query should complete in a few minutes.

SPARQL query to list the graphs in a VIVO

```
SELECT ?g
WHERE
{
    GRAPH ?g {
        ?s ?p ?o .
    }
}
GROUP BY ?g
ORDER BY ?g
```

To list the triples in a named graph, use the query below, substituting the name of the graph you wish to list. Caution: listing the triples in larger graphs may take significant time.

SPARQL query to list the triples in a named graph

```
SELECT ?s ?p ?o
WHERE
{
    GRAPH <http://vitro.mannlib.cornell.edu/filegraph/tbox/sameAs.n3> {
        ?s ?p ?o .
    }
}
```

10.8.3 The graphs used by VIVO

Graph name	Contents
http://vitro.mannlib.cornell.edu/default/asserted-tbox	All ontology triples as asserted
http://vitro.mannlib.cornell.edu/default/inferred-tbox	Triples infered from the asserted ontology triples
http://vitro.mannlib.cornell.edu/default/vitro-kb-2	The main triple store for content
http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata	Triples controlling the application
http://vitro.mannlib.cornell.edu/default/vitro-kb-inf	Triples inferred from the main triple store

Graph name	Contents
http://vitro.mannlib.cornell.edu/filegraph/abox/academicDegree.rdf	Data provided regarding Academic Degrees
http://vitro.mannlib.cornell.edu/filegraph/abox/continents.n3	Data provided regarding the continents
http://vitro.mannlib.cornell.edu/filegraph/abox/dateTimeValuePrecision.owl	Data provided regarding date time precisions
http://vitro.mannlib.cornell.edu/filegraph/abox/documentStatus.owl	Data provided regarding document statuses
http://vitro.mannlib.cornell.edu/filegraph/abox/geopolitical.abox.ver1.1-11-18-11.owl	Data provided regarding geopolitical entities
http://vitro.mannlib.cornell.edu/filegraph/abox/us-states.rdf	Data provided regarding US states and territories
http://vitro.mannlib.cornell.edu/filegraph/abox/validation.n3	Data regarding validation states
http://vitro.mannlib.cornell.edu/filegraph/abox/vocabularySource.n3	Data provided regarding vocabulary sources
http://vitro.mannlib.cornell.edu/filegraph/tbox/ontologies.owl	Descriptions of the ontologies used in VIVO
http://vitro.mannlib.cornell.edu/filegraph/tbox/vitro-0.7.owl	Ontology assertions for the Vitro application, internal
http://vitro.mannlib.cornell.edu/filegraph/tbox/vitroPublic.owl	Ontology assertions for the Vitro application, public. Defines files and file types
http://vitro.mannlib.cornell.edu/filegraph/tbox/vivo.owl	The VIVO Ontology

10.8.4 Notes

1. Graphs named "default" are built and managed by the Vitro application. Graphs names "filegraph" are loaded from files when VIVO starts. Graphs named "filegraph/abox" are data. Graphs named "filegraph/tbox" are ontology.
2. filegraph graphs are named with the name of the file they were loaded from.
3. filegraph files may be in several formats. You will see graphs loaded from files with type n3, owl and rdf.
4. The content in some of the filegraphs may repeat content found in other filegraphs. This does not impact the application.
5. Data you load by placing a file in filegraph/abox will appear as a result of the graph listing query above.

10.9 Ontology Reference

10.9.1 Overview

VIVO uses a collection of ontologies to represent scholarship. The VIVO Ontology provides a set of types (classes) and relationships (properties) to represent researchers and the full context in which they work. Additional ontologies provide context and meaning for attributes and entities defined in the VIVO Ontology and ontologies used by the VIVO Ontology. The VIVO Project maintains the VIVO Ontology. Other ontologies used in VIVO are maintained by the W3C[178] and other groups.

10.9.2 Reference Materials

- VIVO Ontology Domain Definition (see page 447)
- Source ontologies for VIVO (see page 448)
- VIVO Classes (see page 448)
- VIVO Object Properties (see page 450)
- Ontology Diagrams (see page 450)

10.9.3 Issue Tracking

Improvements to the ontologies used in VIVO are treated like all other feature requests and are tracked in the VIVO JIRA issue tracker[179].

---

[178] https://www.w3.org/standards/semanticweb/ontology
[179] https://jira.duraspace.org/projects/VIVO/summary

10.9.4 VIVO Ontology Domain Definition

The VIVO ontology [1] (see page 0) is used to represent the expertise of people engaged in the creation, transmission, and preservation of knowledge and creative works. The VIVO ontology (hereafter referred to as "the ontology") represents expertise by describing the activities and accomplishments of people in terms of their relationships to particular artifacts of the work, resources they use, institutions that employ them, and other indicators. The ontology is independent of knowledge or creative domain. The ontology supports the identification, evaluation, and impact assessment of individual people and groups of people, as well as identification and reuse of the works of the people.

Ontology competency questions [2] (see page 0)

1. Who are the people with expertise in subject w?
2. What people are available to work on activity x?
3. What is the scholarly output of institution y over time period t?
4. Which people have created or have proficiency in the use of resource z?
5. What patents, papers, awards, grants, datasets, art work, performances, credentials, or other evidence of expertise does a person have?
6. What is the scope of the expertise of an person? How is it limited in time, geography, or domain?
7. What is the provenance of the evidence regarding expertise?

Consequences and observations

1. The ontology requires precise identification of people, works, and the relationships between people and works.
2. There are many forms of evidence for expertise. These forms have common elements (authored materials, grants obtained, courses taught, education and training, service to the profession), while other forms may be domain specific, such as required by the Arts, Sciences, Education, Engineering, Law, Business, Agriculture, or Medicine [3] (see page 0) . The ontology is expected to be extended by related modules to represent forms of evidence of expertise that are domain or discipline specific. These forms need not appear in the VIVO ontology.
3. The ontology requires precise representation of entities that occur in the creation, transmission, and preservation of knowledge and creative works, including, but not limited to: institutions, geographical locations, time intervals and time points, and entities and properties forming the precise descriptions of the evidence for expertise.
4. Expertise is not limited to proficiency in particular activities, such as research. The creation, transmission, and preservation of knowledge and creative works is not limited to particular activities, nor to particular types of employment (faculty) nor to particular types of institutions (academic), nor to particular regions of the world.

10.9.5 Source ontologies for VIVO

10.9.5.1 Background

Source ontologies may be imported in their entirety or included selectively by importing terms and the terms required to support the required terms.

10.9.5.2 Ontologies Used in the VIVO Ontology

The VIVO Ontology leverages the following ontologies in a unified, semantic structure:

- eagle-i Resource Ontology (ERO) – http://www.obofoundry.org/ontology/ero.html
- Basic Formal Ontology (BFO) – http://www.obofoundry.org/ontology/bfo.html
- Bibliographic Ontology (BIBO) – http://bibliontology.com/
- Event Ontology – http://motools.sourceforge.net/event/event.html
- Friend of a Friend (FOAF) – http://www.foaf-project.org/
- Gene Ontology (GO) – http://obofoundry.org/ontology/go.html
- Geopolitical.owl[180], from the U.N. Food and Agriculture Organization
- Information Artifact Ontology (IAO) – http://www.obofoundry.org/ontology/iao.html
- Ontology for Biomedical Investigations (OBI) – http://www.obofoundry.org/ontology/obi.html
- Ontology of Clinical Research (OCRe) – http://code.google.com/p/ontology-of-clinical-research/
- Relations Ontology (RO) – http://www.obofoundry.org/ontology/ro.html
- Software Ontology (SWO) – http://www.obofoundry.org/ontology/swo.html
- SKOS (Simple Knowledge Organization System) – http://www.w3.org/2004/02/skos/
- vCard – http://www.w3.org/TR/vcard-rdf/
- SPAR ontologies, including FABIO, CiTO, and C4O: https://purl.org/spar/fabio

10.9.6 VIVO Classes

10.9.6.1 Overview

VIVO uses a large number of classes from several different ontologies to represent scholarship. See Source ontologies for VIVO (see page 448). The classes and their ontologies are shown in the figure below. You may have additional classes as a result of local extensions.

10.9.6.2 Finding the Classes in your VIVO

To find the classes in your VIVO, you can use the SPARQL query below.

---

[180] http://www.fao.org/countryprofiles/geopol_v10/ontologies/geopolitical.owl.html

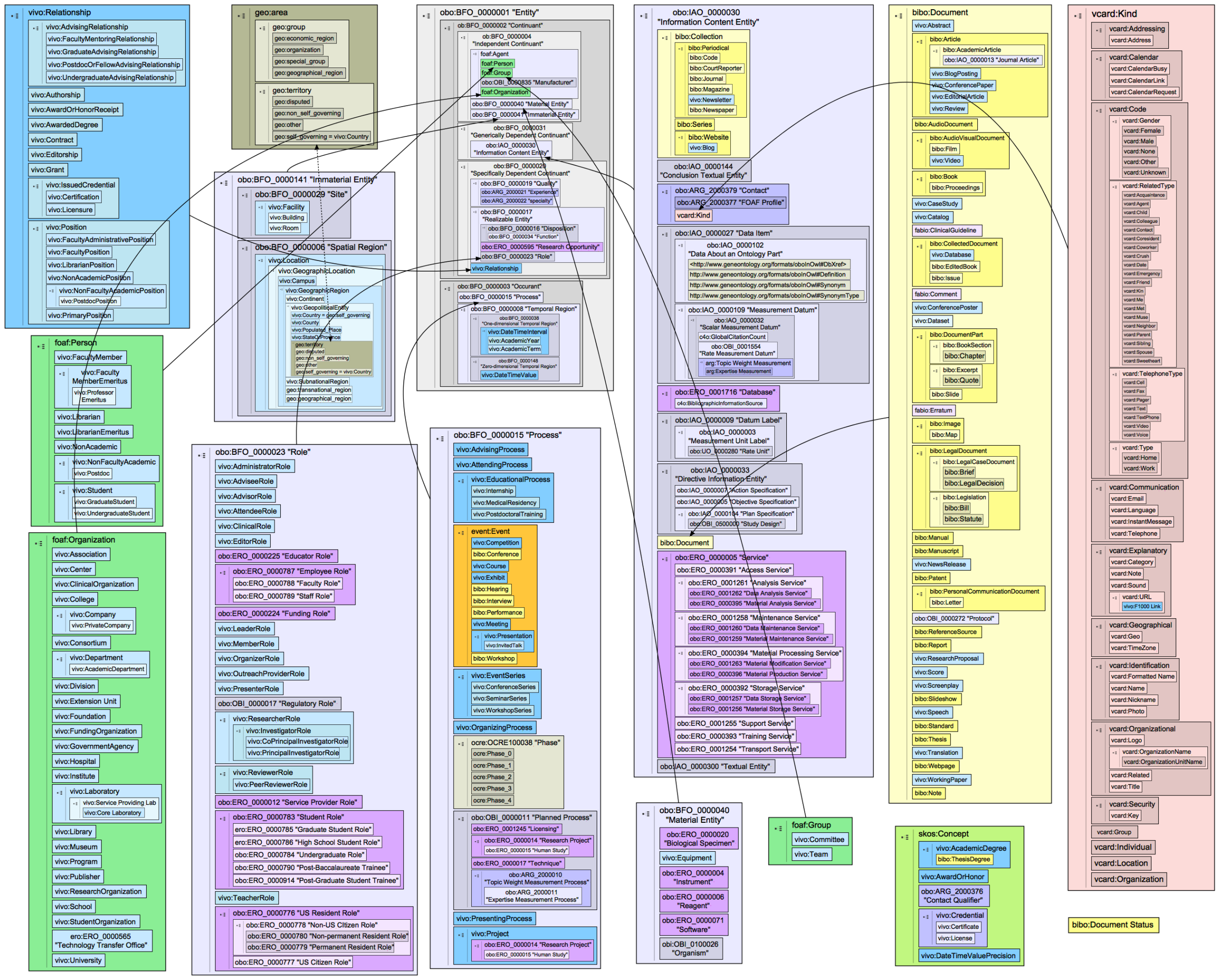
```
SELECT ?s ?label
WHERE
{
   ?s a owl:Class .
   FILTER(regex(?s, "http"))
   ?s rdfs:label ?label .
}
ORDER BY ?s
```

10.9.6.3 VIVO Classes

All classes delivered with VIVO should be included in the diagram below. Classes in the respective ontologies, but not delivered in VIVO, are not included. For figures related to the ontologies on which VIVO is based, see the corresponding ontology projects.

VIVO Classes

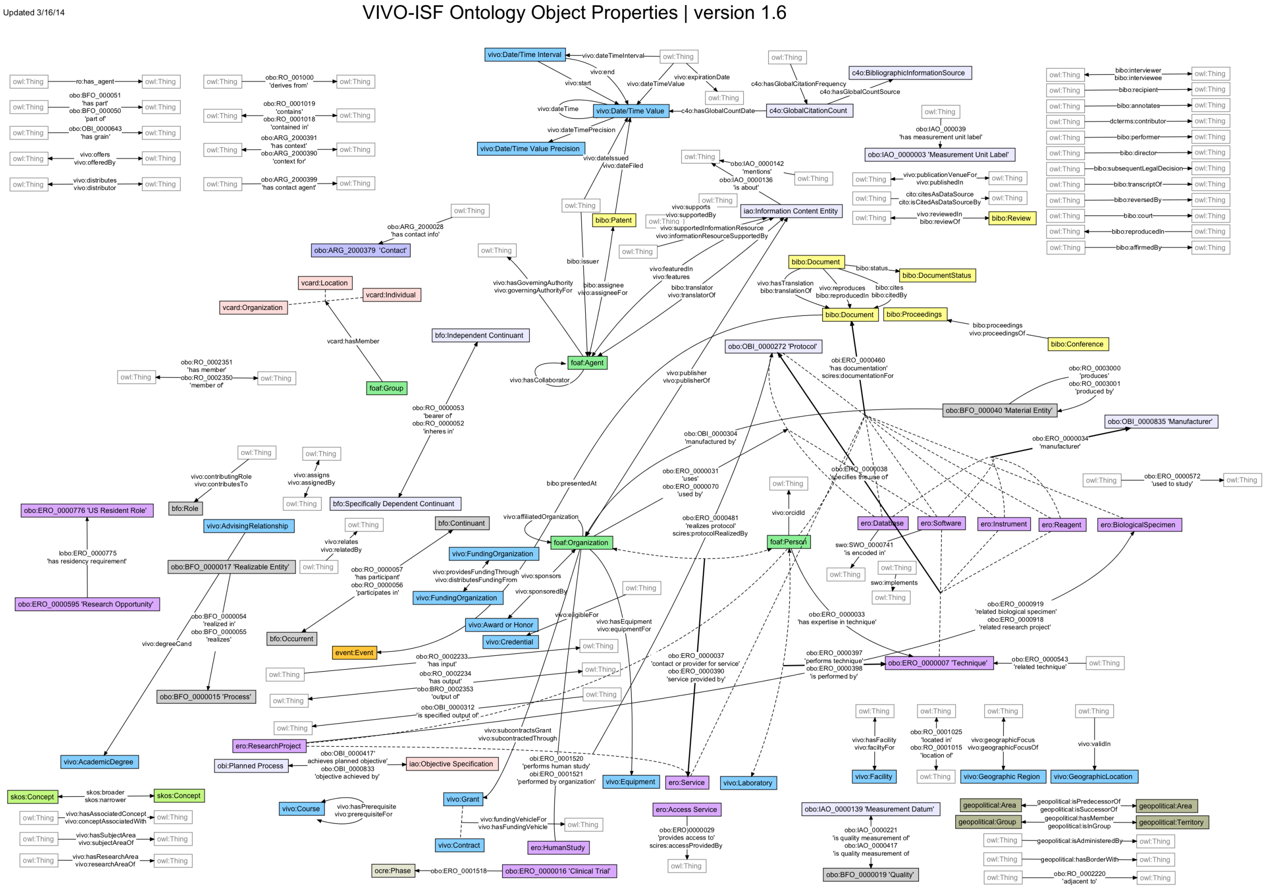
12 October 2016


vivo:Relationship
geo:area
obo:BFO_0000001 "Entity"
obo:IAO_0000030 "Information Content Entity"
bibo:Document
vcard:Kind
foaf:Person
foaf:Organization
obo:BFO_0000141 "Immaterial Entity"
obo:BFO_0000023 "Role"
obo:BFO_0000015 "Process"
obo:BFO_0000040 "Material Entity"
foaf:Group
skos:Concept
bibo:Document Status

10.9.7 VIVO Object Properties

The diagram shows object properties used in the VIVO ontology.

Faux properties are a VIVO software feature that allows for commonly used properties used in the context of specific domain and range classes to have familiar names.



Updated 3/16/14

VIVO-ISF Ontology Object Properties | version 1.6

vivo:Date/Time Interval
vivo:Date/Time Value
vivo:Date/Time Value Precision
c4o:BibliographicInformationSource
c4o:GlobalCitationCount
obo:IAO_0000003 'Measurement Unit Label'
iao:Information Content Entity
bibo:Patent
bibo:Review
bibo:Document
bibo:DocumentStatus
bibo:Proceedings
bibo:Conference
obo:ARG_2000379 'Contact'
vcard:Location
vcard:Individual
vcard:Organization
bfo:Independent Continuant
foaf:Agent
foaf:Group
obo:OBI_0000272 'Protocol'
obo:BFO_000040 'Material Entity'
obo:OBI_0000835 'Manufacturer'
obo:ERO_0000776 'US Resident Role'
bfo:Role
vivo:AdvisingRelationship
bfo:Specifically Dependent Continuant
bfo:Continuant
obo:BFO_0000017 'Realizable Entity'
obo:ERO_0000595 'Research Opportunity'
vivo:FundingOrganization
foaf:Organization
foaf:Person
ero:Database
ero:Software
ero:Instrument
ero:Reagent
ero:BiologicalSpecimen
bfo:Occurrent
event:Event
vivo:Award or Honor
vivo:Credential
obo:ERO_0000007 'Technique'
obo:BFO_0000015 'Process'
ero:ResearchProject
obi:Planned Process
ero:Objective Specification
vivo:AcademicDegree
skos:Concept
vivo:Equipment
ero:Service
vivo:Laboratory
vivo:Facility
vivo:Geographic Region
vivo:GeographicLocation
vivo:Course
vivo:Grant
vivo:Contract
ero:HumanStudy
ero:Access Service
obo:IAO_0000139 'Measurement Datum'
obo:BFO_0000019 'Quality'
geopolitical:Area
geopolitical:Group
geopolitical:Territory
ocre:Phase
obo:ERO_0000016 'Clinical Trial'
owl:Thing

10.9.8 Ontology Diagrams

These diagrams shows the relationships between entities in VIVO. Diagrams for the primary entities of scholarship such as people and publications have diagrams centered on the primary entity.

Diagrams are drawn using VUE (Visual Understanding of the Environment), open source software from Tufts University. You can learn more about VUE at the VUE website[181]. VUE versions of each diagram are attached to the corresponding ontology diagram page.

These diagrams focus on the entity of interest. Related entities are not shown in complete detail.

---

[181] http://vue.tufts.edu

These diagrams focus on common attributes and relationships. They are not comprehensive. Additional attributes and relationships are available. See the VIVO interface, use SPARQL queries, and examine the ontology to discover additional attributes and relationships.

- Organization Model (see page 451)
- Concept Model (see page 453)
- DateTimeValue and DateTimeInterval Models (see page 454)
- Journal Model (see page 457)
- Person Model (see page 458)
- Teaching Model (see page 459)
- Publication Model (see page 461)
- Grant Model (see page 462)
- Education and Training Model (see page 469)
- Advising Model (see page 471)
- Award Model (see page 473)
- Membership Model (see page 474)
- Ontology Diagram Legend (see page 475)
- Credential Model (see page 475)

10.9.8.1 Organization Model

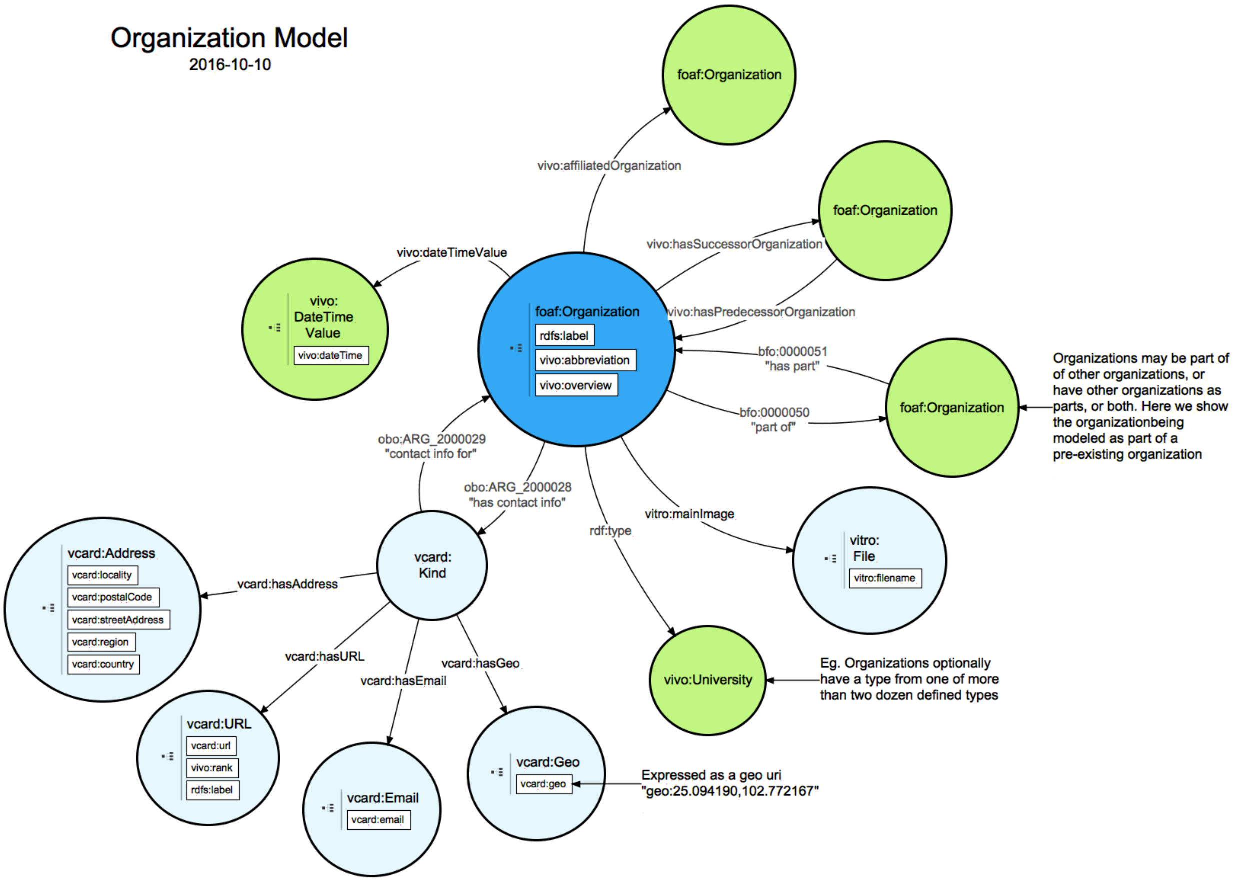
10.9.8.1.1 Notes

1. Organizations in VIVO are entities with `rdf:type foaf:Organization`. `vivo:overview` is used to provide a text description of the organization typically displayed on its profile page. A vcard is used to record contact information, URLs and geolocation.
2. VIVO provides a controlled vocabulary of organization types as `rdfs:subClassOf foaf:Organization.` To create a list of the available organization types, use the SPARQL query below:

```
SELECT ?s
WHERE
{
    ?s rdfs:subClassOf foaf:Organization .
}
```

3. Organizations may have relationships to other organizations. The "part of" relationship describes an organization as part of another in a hierarchical sense. For example, the History Department may be part of a College of Liberal Arts. The "successor" relationship describes an organization which no longer exists, and for which a successor organization now exists. The "affiliatedOrganization" organization describes an organization affiliated with the primary organization. The relationship is not symmetric, that is, the inverse is not inferred by the Inferencer. Assert the reverse affiliation as needed.

4. Many other attributes and relationships are available for organizations. The model shown here is typical for VIVO implementations.


Organization Model
2016-10-10

foaf:Organization
vivo:affiliatedOrganization
foaf:Organization
vivo:hasSuccessorOrganization
vivo:dateTimeValue
vivo:
DateTime
Value
vivo:dateTime
foaf:Organization
rdfs:label
vivo:abbreviation
vivo:overview
vivo:hasPredecessorOrganization
bfo:0000051
"has part"
bfo:0000050
"part of"
foaf:Organization
obo:ARG_2000029
"contact info for"
obo:ARG_2000028
"has contact info"
vitro:mainImage
rdf:type
vitro:
File
vitro:filename
vcard:Address
vcard:locality
vcard:postalCode
vcard:streetAddress
vcard:region
vcard:country
vcard:hasAddress
vcard:
Kind
vcard:hasURL
vcard:hasGeo
vcard:hasEmail
vivo:University
vcard:URL
vcard:url
vivo:rank
rdfs:label
vcard:Email
vcard:email
vcard:Geo
vcard:geo

Organizations may be part of of other organizations, or have other organizations as parts, or both. Here we show the organizationbeing modeled as part of a pre-existing organization

Eg. Organizations optionally have a type from one of more than two dozen defined types

Expressed as a geo uri "geo:25.094190,102.772167"

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.2 Concept Model

10.9.8.2.1 Notes

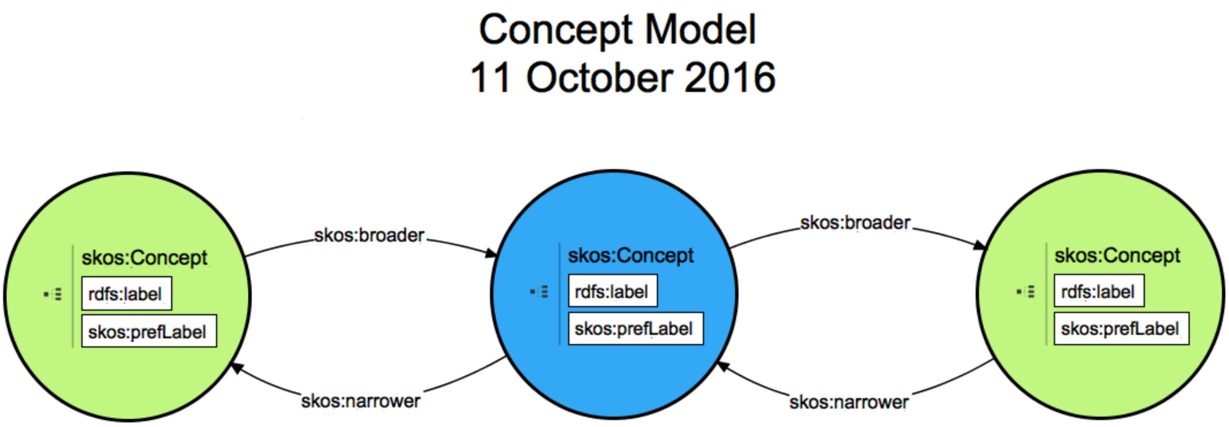
1. Concepts in VIVO are modeled using the SKOS (Simple Knowledge Organization System) ontology. SKOS is quite simple, and is a good place to start for those learning about ontologies, and how VIVO uses ontologies to represent information as triples in RDF. See The SKOS Primer[182], a readable introduction to SKOS and how it is represented in RDF.
2. A concept is typically represented in VIVO as two triples, one declaring the URI of the concept as a skos:Concept, and one providing a text label for the concept. A third triple may use the skos:prefLabel to repeat the text label for those applications expecting the concept to have a preferred label. The triples might look like those below:

```
<http://vivo.myschool.edu> rdf:type skos:Concept .
<http://vivo.myschool.edu> rdfs:label "Molecular Biology"^^@en .
<http://vivo.myschool.edu> skos:prefLabel "Molecular Biology"^^@en .
```

3. Concepts are used throughout VIVO to indicate research and subject areas for people and other entities.

---

[182] https://www.w3.org/TR/skos-primer/


Concept Model
11 October 2016
skos:Concept
rdfs:label
skos:prefLabel
skos:broader
skos:narrower
skos:Concept
rdfs:label
skos:prefLabel
skos:broader
skos:narrower
skos:Concept
rdfs:label
skos:prefLabel

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.3 DateTimeValue and DateTimeInterval Models

10.9.8.3.1 Notes

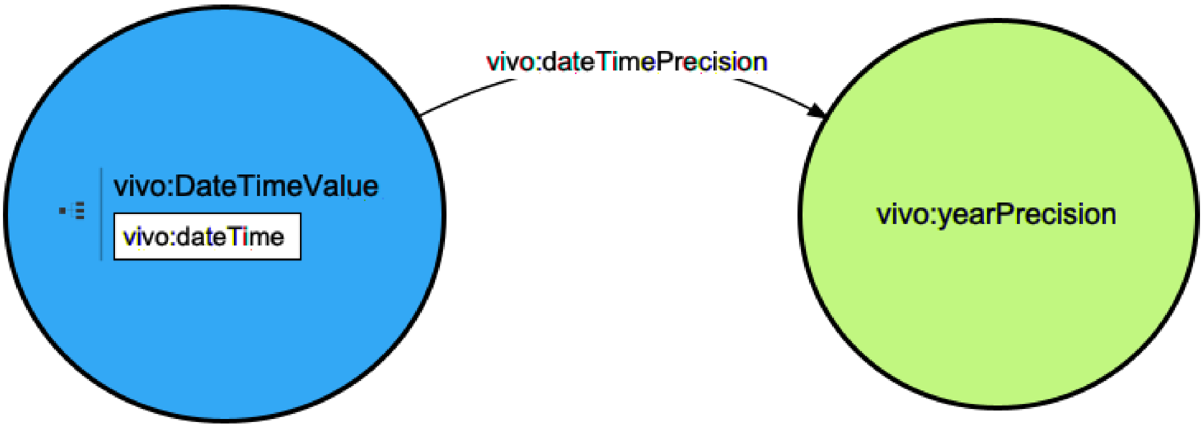
1. VIVO uses DateTimeValue and DateTimeInterval to model dates and datetimes. These are objects, not literal values. The object models are simple (see below). VIVO DateTimeValue supports the concept of a precision, which indicates whether a particular DateTimeValue is accurate to the day, or

perhaps only to the month, or perhaps only to the year. Precision is an important idea – publication dates, for example, are often known only to year precision, and sometimes to year and month.

2. The model indicates that creating a DateTimeValue requires three triples – one to specify the type, one to specify the literal value of the datetime, and one to indicate the precision.

```
<http://vivo.myschool.edu/individual/n123> rdf:type vivo:DateTimeValue .
<http://vivo.myschool.edu/individual/n123> vivo:dateTime
"2010-11-12T12:00:00"^^xsd:datetime .
<http://vivo.myschool.edu/individual/n123> vivo:dateTimePrecision
vivo:yearPrecision .
```

3. VIVO provides the precisions shown below:
    - `<http://vivoweb.org/ontology/core#yearMonthDayTimePrecision>`
    - `<http://vivoweb.org/ontology/core#yearMonthPrecision>`
    - `<http://vivoweb.org/ontology/core#yearPrecision>`
    - `<http://vivoweb.org/ontology/core#yearMonthDayPrecision>`



Ontology Diagram Legend

Dark blue – the entity being modeled

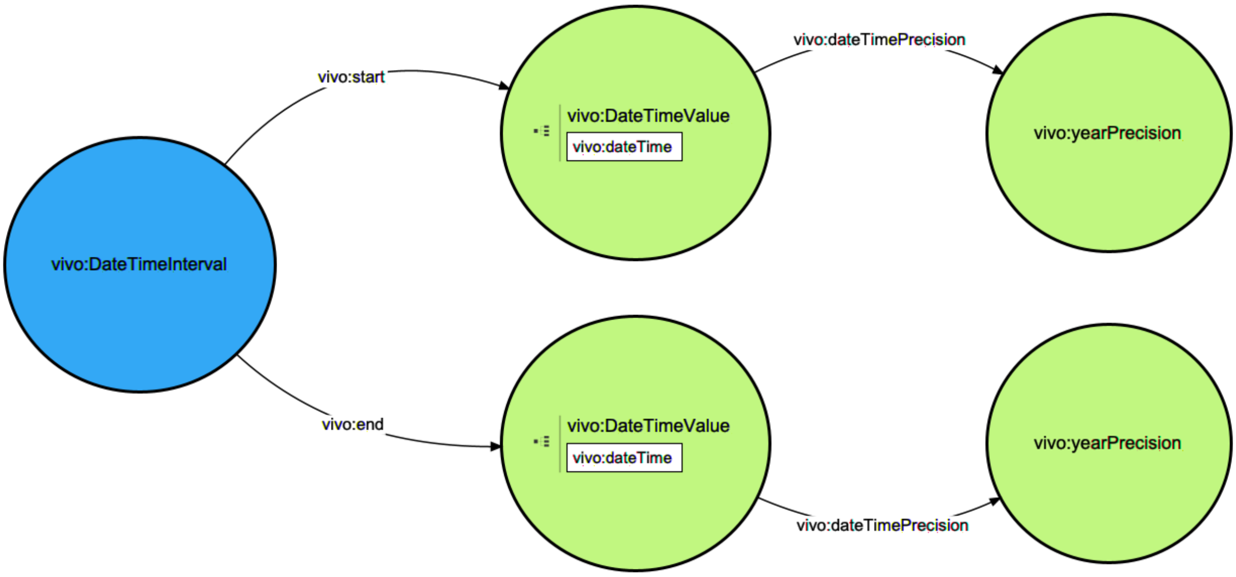
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.3.2 DateTimeInterval

The DateTimeInterval is an entity that references one or two DateTimeValues. Either reference could be missing. An interval might have a start date and no end date, for example. To create a DateTimeValue with a start and end takes the statements below, where the start and end objects exist and have the URI as shown.

```
<http://vivo.mydomain.edu/individual/n456> rdf:type vivo:DateTimeInterval .
<http://vivo.mydomain.edu/individual/n456> vivo:start <http://vivo.mydomain.edu/
individual/n123> .
<http://vivo.mydomain.edu/individual/n456> vivo:end <http://vivo.mydomain.edu/
individual/n124> .
```


vivo:DateTimeInterval
vivo:start
vivo:end
vivo:DateTimeValue
vivo:dateTime
vivo:DateTimeValue
vivo:dateTime
vivo:dateTimePrecision
vivo:dateTimePrecision
vivo:yearPrecision
vivo:yearPrecision

Ontology Diagram Legend

Dark blue – the entity being modeled

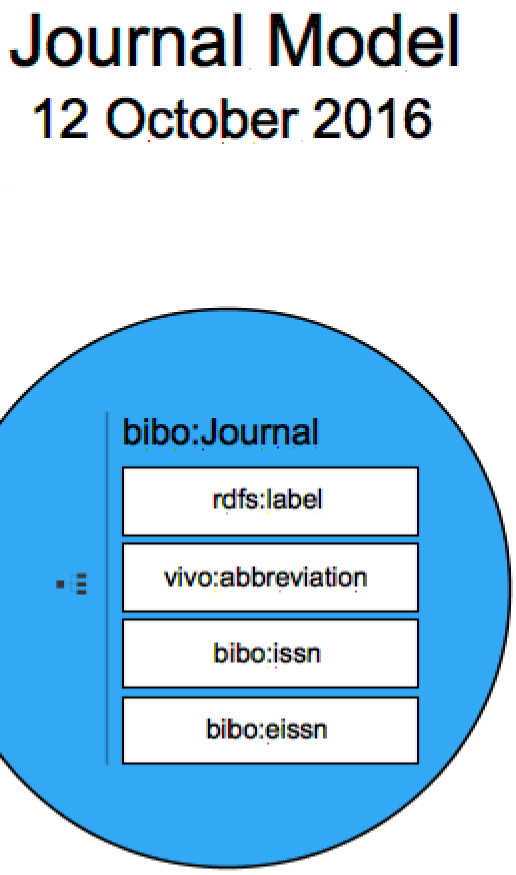
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.4 Journal Model

10.9.8.4.1 Notes

1. A journal in VIVO is an entity of type `bibo:Journal`.
2. The journal has a series of attributes, all are literals. Journal entities are quite simple.


Journal Model
12 October 2016
bibo:Journal
rdfs:label
vivo:abbreviation
bibo:issn
bibo:eissn

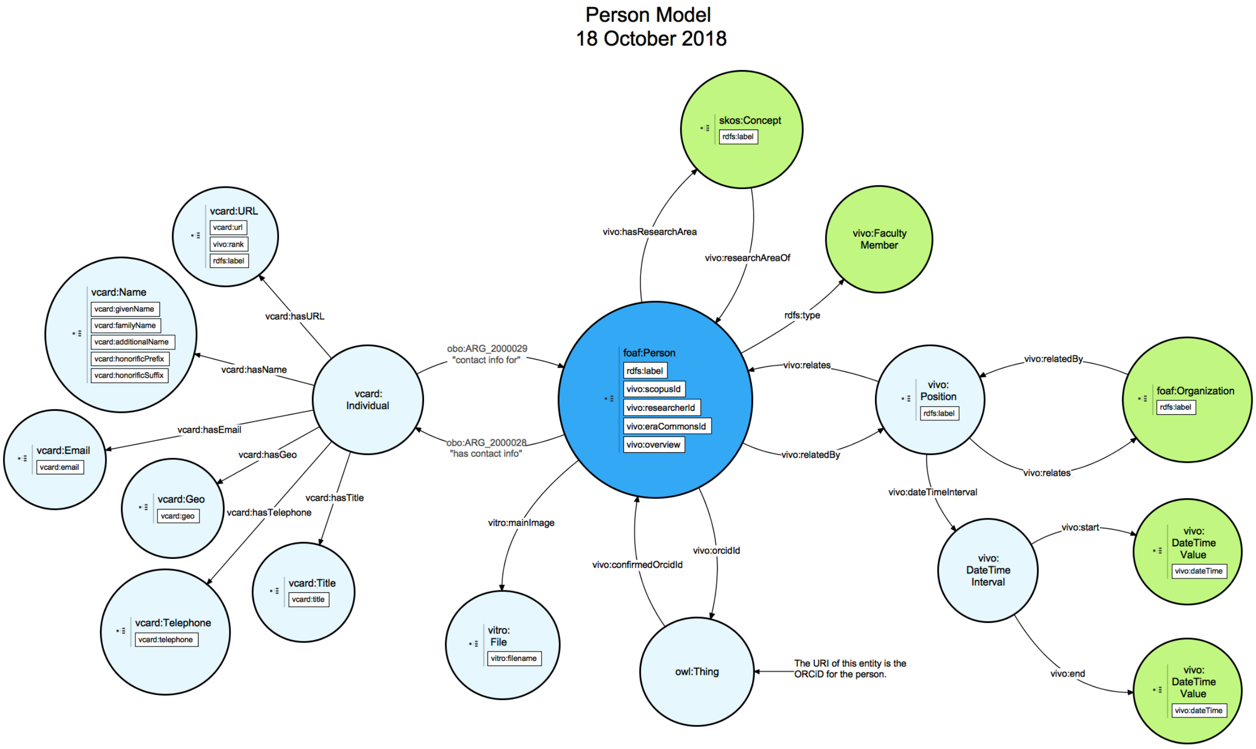
Ontology Diagram Legend
Dark blue – the entity being modeled Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.5 Person Model

10.9.8.5.1 Notes

1. The Person entity has a large collection of literal values. The most common are shown in the Person entity below.
2. People are associated with research areas represented by skos:Concept entities. See Concept Model (see page 453).
3. Positions are relationships between a person and an organization. See Organization Model (see page 451) for detail regarding representation of organizations. The position may have an associated dateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 454) for details regarding the representation of these entities.
4. The ORCiD of a person is represented as an entity. The URI of the entity is the ORCiD of the person. See Managing Person Identifiers (see page 109) for additional details.
5. The photo of a person is stored in a file and referenced using triples associated with the person. See Image storage (see page 425)
6. Vcards are used to store contact information, name parts, URLs, and geolocation. The general pattern is that a person has a vcard, the vcard has a an intermediate related to the type of information to be stored, and the intermediate has references to literal values.
7. Additional details regarding the person – teaching (see page 459), grants (see page 462), publications (see page 461), advising (see page 471), educational training (see page 469), awards (see page 473), memberships (see page 474) – are shown in their respective models.


Person Model
18 October 2018
skos:Concept
rdfs:label
vivo:hasResearchArea
vivo:researchAreaOf
vivo:Faculty Member
rdfs:type
vcard:URL
vcard:url
vivo:rank
rdfs:label
vcard:hasURL
vcard:Name
vcard:givenName
vcard:familyName
vcard:additionalName
vcard:honorificPrefix
vcard:honorificSuffix
vcard:hasName
vcard: Individual
obo:ARG_2000029
"contact info for"
obo:ARG_2000028
"has contact info"
foaf:Person
rdfs:label
vivo:scopusId
vivo:researcherId
vivo:eraCommonsId
vivo:overview
vivo:relates
vivo:relatedBy
vivo: Position
rdfs:label
vivo:relatedBy
vivo:relates
foaf:Organization
rdfs:label
vcard:Email
vcard:email
vcard:hasEmail
vcard:hasGeo
vcard:Geo
vcard:geo
vcard:hasTelephone
vcard:hasTitle
vcard:Title
vcard:title
vcard:Telephone
vcard:telephone
vitro:mainImage
vitro: File
vitro:filename
vivo:confirmedOrcidId
vivo:orcidId
owl:Thing
The URI of this entity is the ORCiD for the person.
vivo:dateTimeInterval
vivo: DateTime Interval
vivo:start
vivo:end
vivo: DateTime Value
vivo:dateTime
vivo: DateTime Value
vivo:dateTime

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

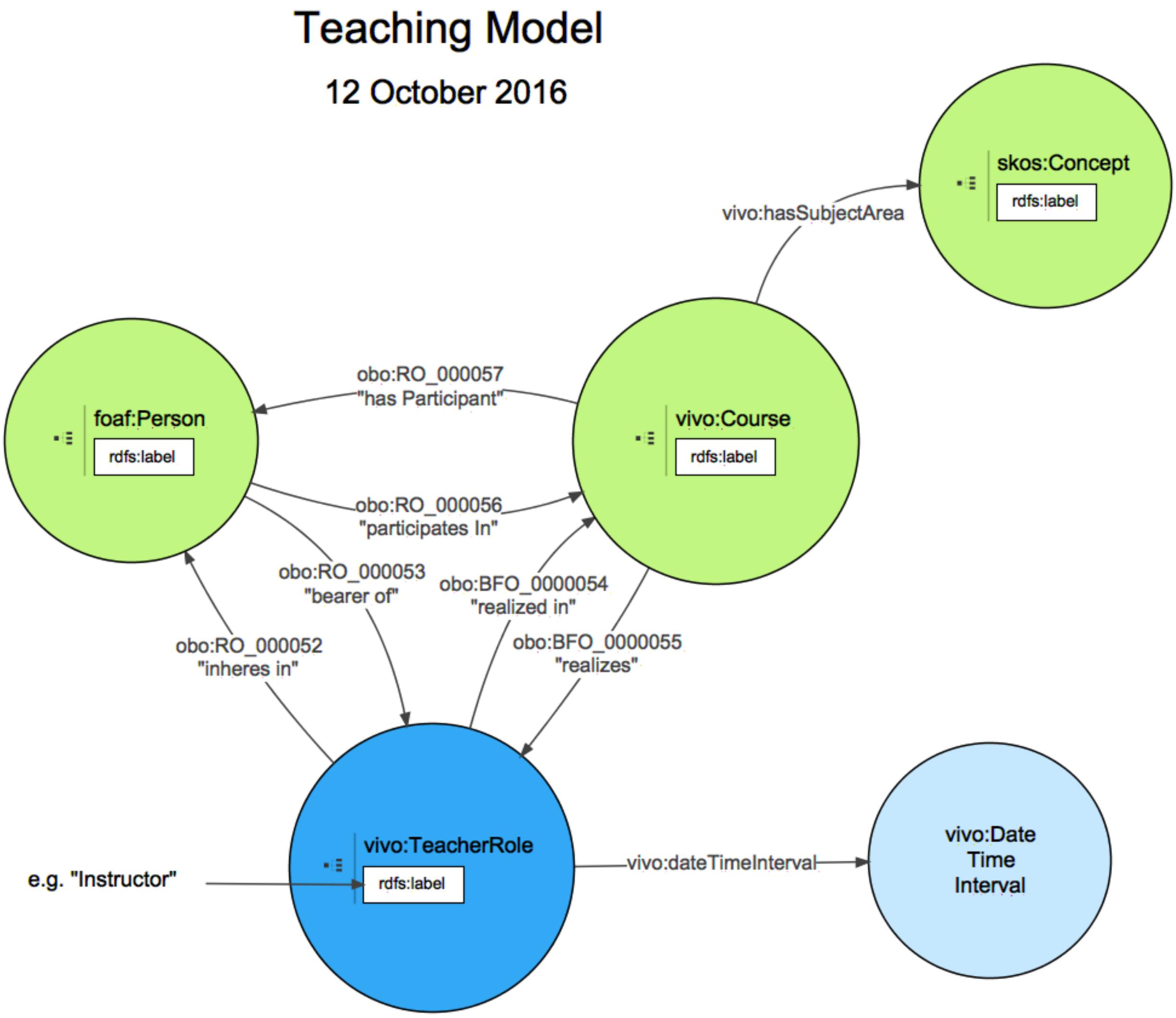
Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.6 Teaching Model

10.9.8.6.1 Notes

1. Teaching is represented as a time limited role associating a person with a course.
2. The course may have optional concepts indicating subject area(s). See Concept Model (see page 453) for details.

3. The role typically has a DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 454) for details.
4. The Role may have a label such as "Instructor" or "Team Lead" or other to further indicate the nature of the instructor's role.
5. The instructor is a person. See Person Model (see page 458) for details.
6. Any number of instructors may each have a role in a course. Each has their own role.

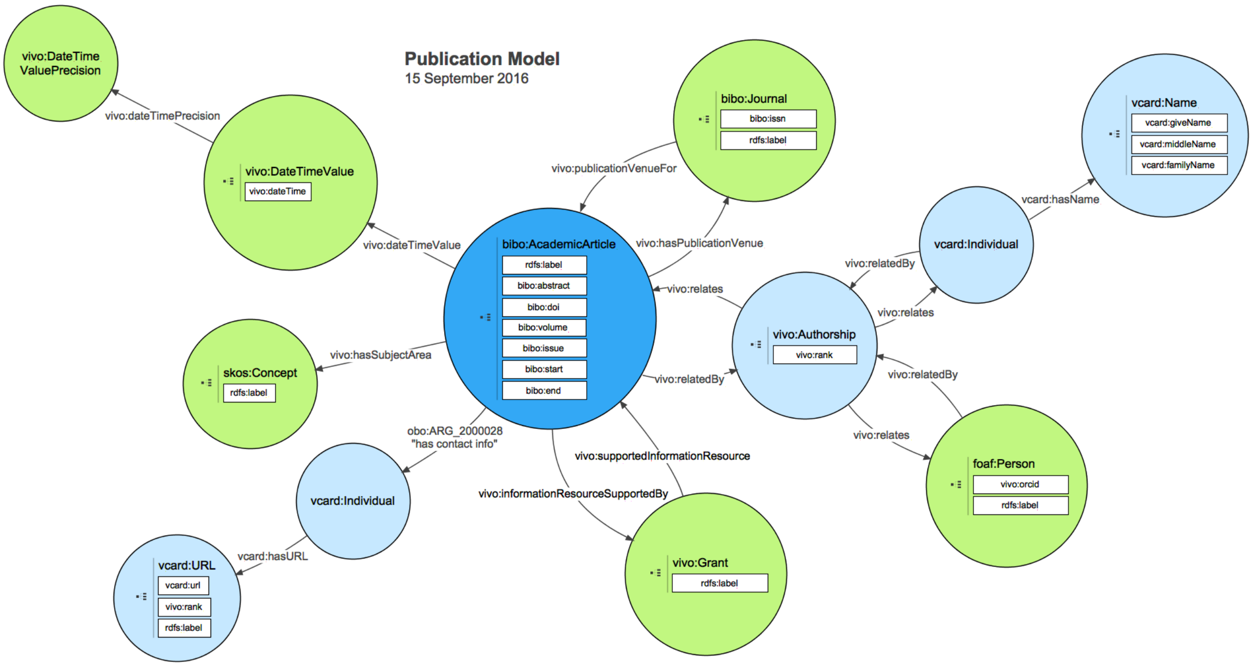

Teaching Model
12 October 2016
skos:Concept
rdfs:label
vivo:hasSubjectArea
foaf:Person
rdfs:label
vivo:Course
rdfs:label
obo:RO_000057
"has Participant"
obo:RO_000056
"participates In"
obo:RO_000053
"bearer of"
obo:BFO_0000054
"realized in"
obo:RO_000052
"inheres in"
obo:BFO_0000055
"realizes"
e.g. "Instructor"
vivo:TeacherRole
rdfs:label
vivo:dateTimeInterval
vivo:Date Time Interval

Ontology Diagram Legend
Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.7 Publication Model


Publication Model
15 September 2016
vivo:DateTime ValuePrecision
vivo:dateTimePrecision
vivo:DateTimeValue
vivo:dateTime
vivo:dateTimeValue
bibo:Journal
bibo:issn
rdfs:label
vivo:publicationVenueFor
vivo:hasPublicationVenue
vcard:Name
vcard:giveName
vcard:middleName
vcard:familyName
vcard:hasName
vcard:Individual
bibo:AcademicArticle
rdfs:label
bibo:abstract
bibo:doi
bibo:volume
bibo:issue
bibo:start
bibo:end
vivo:relates
vivo:relatedBy
vivo:relates
vivo:Authorship
vivo:rank
vivo:relatedBy
vivo:relatedBy
vivo:relates
skos:Concept
rdfs:label
vivo:hasSubjectArea
obo:ARG_2000028
"has contact info"
vivo:supportedInformationResource
vivo:informationResourceSupportedBy
foaf:Person
vivo:orcid
rdfs:label
vcard:Individual
vcard:hasURL
vcard:URL
vcard:url
vivo:rank
rdfs:label
vivo:Grant
rdfs:label

Ontology Diagram Legend

Dark blue – the entity being modeled

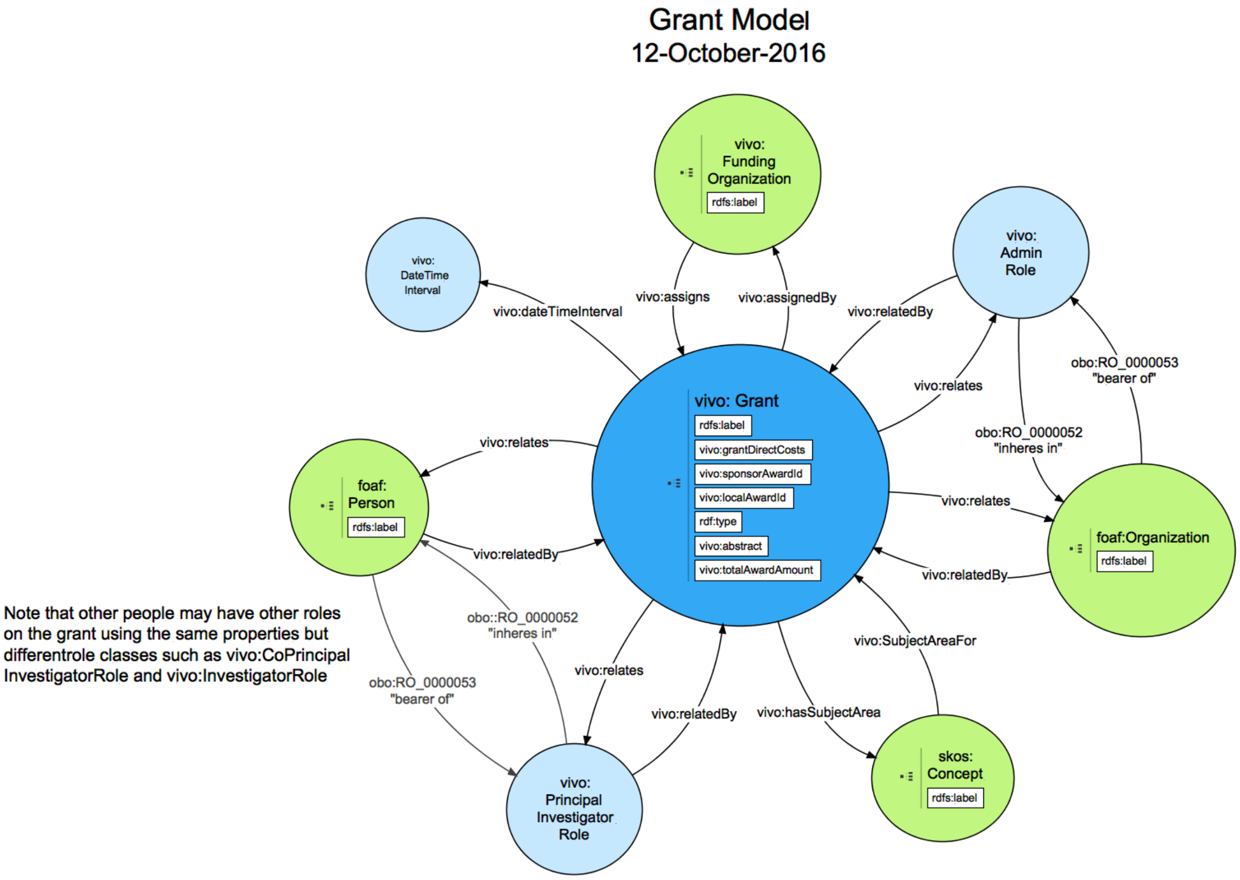
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.8 Grant Model

10.9.8.8.1 Notes

1. The Grant entity has attributes to record funding amounts, label, abstract, local award ID (the ID as assigned by the administering organization), and sponsorAwardID (the ID of the grant as assigned by the funding organization)
2. The FundingOrganization is related to the grant through vivo:assigns and vivo:assignedBy. For additional details regarding modeling organizations, see Organization Model (see page 451)
3. An organization, often an academic department, typically has a role in administering the grant. This is modeling using an AdminRole which associated the grant, the role and the organization administering the grant.
4. The grant may have one or more subject areas, represented as skos:Concept. See Concept Model (see page 453).
5. One or more people will be associated with the grant through roles. There will be one role for each person. The role associates the person with the grant. For additional detail regarding the modeling of people, see Person Model (see page 458)
6. The grant has an associate dateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 454)


Grant Model
12-October-2016
vivo: Funding Organization
rdfs:label
vivo: DateTime Interval
vivo: Admin Role
vivo:dateTimeInterval
vivo:assigns
vivo:assignedBy
vivo:relatedBy
obo:RO_0000053 "bearer of"
vivo:relates
vivo: Grant
rdfs:label
vivo:grantDirectCosts
vivo:sponsorAwardId
vivo:localAwardId
rdf:type
vivo:abstract
vivo:totalAwardAmount
obo:RO_0000052 "inheres in"
vivo:relates
foaf: Person
rdfs:label
vivo:relates
foaf:Organization
rdfs:label
vivo:relatedBy
vivo:relatedBy
Note that other people may have other roles on the grant using the same properties but differentrole classes such as vivo:CoPrincipal InvestigatorRole and vivo:InvestigatorRole
obo::RO_0000052 "inheres in"
vivo:SubjectAreaFor
vivo:relates
obo:RO_0000053 "bearer of"
vivo:relatedBy
vivo:hasSubjectArea
skos: Concept
rdfs:label
vivo: Principal Investigator Role

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.8.2 Worked ontology example using Person, Role, and Project instead of Grant

A question has come up in the VIVO community about using Project instead of Grants – when a VIVO institution may not receive grants but does want to track projects.

In the VIVO-ISF ontology, a Grant is a subclass of vivo:Relationship, since it represents the agreement between a funding organization and a receiving organization, with the investigator roles usually also specified.

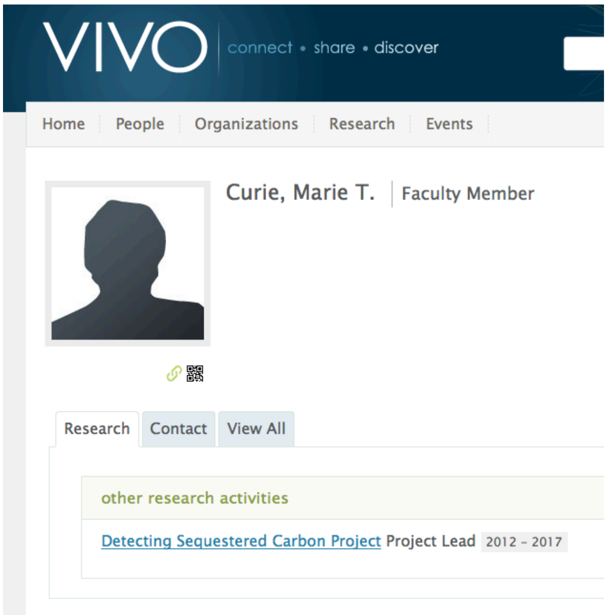
A Project, however, is a subclass of Project, which in turn is a subclass of bfo:Process. The project is the activity undertaken or the investigation, not just the agreement.

The properties used are therefore slightly different to connect a Person, Role, and Project vs. a Person, Role, and Grant, as indicated on the Grant Model (see page 462) page.

10.9.8.8.2.1 Example

We have a researcher, Marie Curie, who has the Project Lead role on a Project. In the VIVO front end display, there appears to be a direct relationship between the person and the project, and an inverse relationship in return. The role and date information appear as modifiers to the direct relationship, but are maintained through the ontology as an intermediate Role object bearing the title of the role ("Project Lead") and the date range.

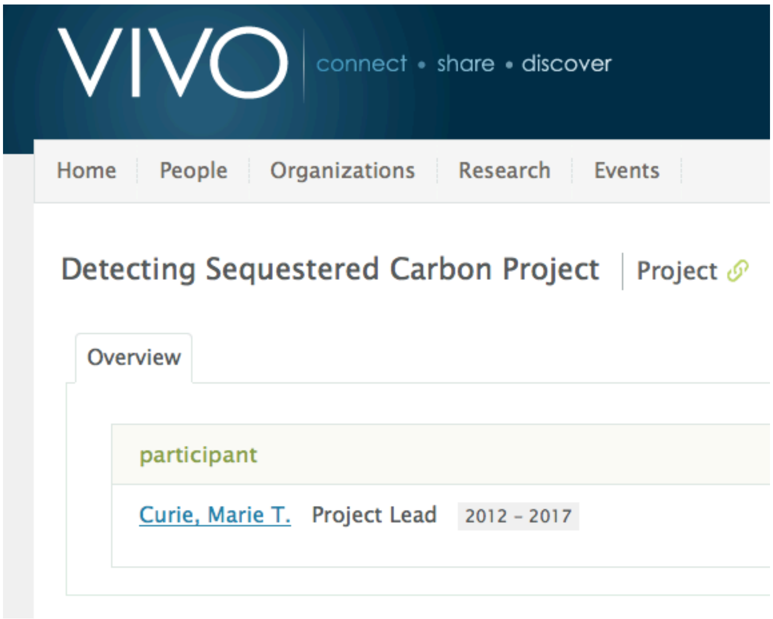
Public display view


VIVO
connect • share • discover
Home
People
Organizations
Research
Events
Research
Contact
View All

Curie, Marie T. | Faculty Member

other research activities

Detecting Sequestered Carbon Project Project Lead 2012 - 2017


VIVO
connect • share • discover
Home
People
Organizations
Research
Events
Overview

Detecting Sequestered Carbon Project | Project

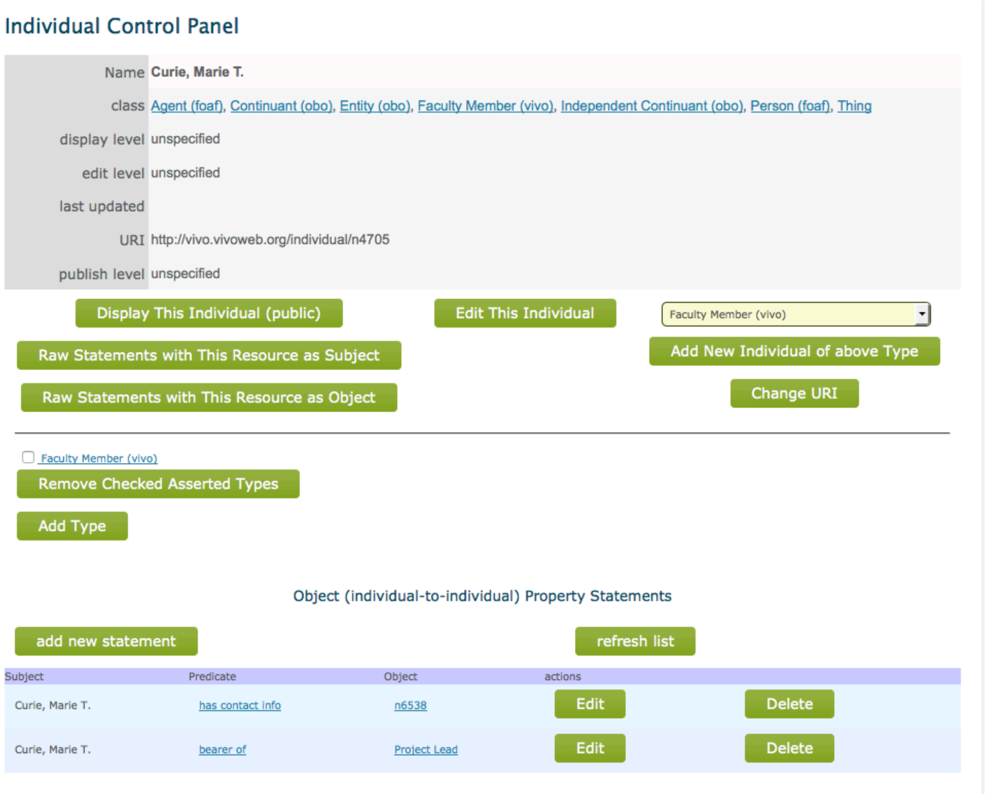
participant

Curie, Marie T. Project Lead 2012 - 2017

Site admin view

This can be seen more clearly in the back-end editors view (when logged in with Site Admin privileges):

Note in the listing of object property statements at the bottom of the image that the Person has a "bearer of" relationship (http://purl.obolibrary.org/obo/RO_0000053) to the Role – and no direct relationship to the Project.



Individual Control Panel

Name	**Curie, Marie T.**
class	Agent (foaf), Continuant (obo), Entity (obo), Faculty Member (vivo), Independent Continuant (obo), Person (foaf), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n4705
publish level	unspecified

Faculty Member (vivo)

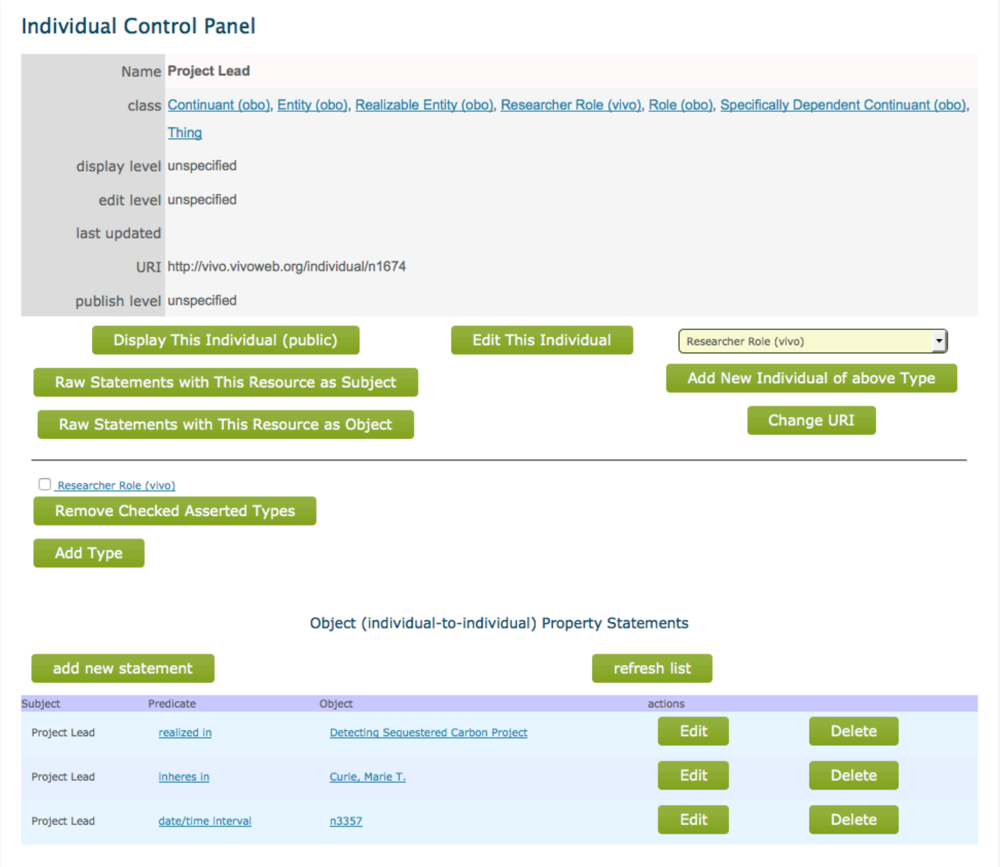
☐ Faculty Member (vivo)

Object (individual-to-individual) Property Statements

Subject	Predicate	Object	actions	
Curie, Marie T.	has contact info	n6538	Edit	Delete
Curie, Marie T.	bearer of	Project Lead	Edit	Delete

Display This Individual (public)
Edit This Individual
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
Remove Checked Asserted Types
Add Type
add new statement
refresh list

On the intermediate Role page, the relationships in both directions may be seen: the Role "inheres in" (http://purl.obolibrary.org/obo/RO_0000052) the Person and is "realized in" (http://purl.obolibrary.org/obo/BFO_0000054) the Project.



Individual Control Panel

Name	**Project Lead**
class	Continuant (obo), Entity (obo), Realizable Entity (obo), Researcher Role (vivo), Role (obo), Specifically Dependent Continuant (obo), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n1674
publish level	unspecified

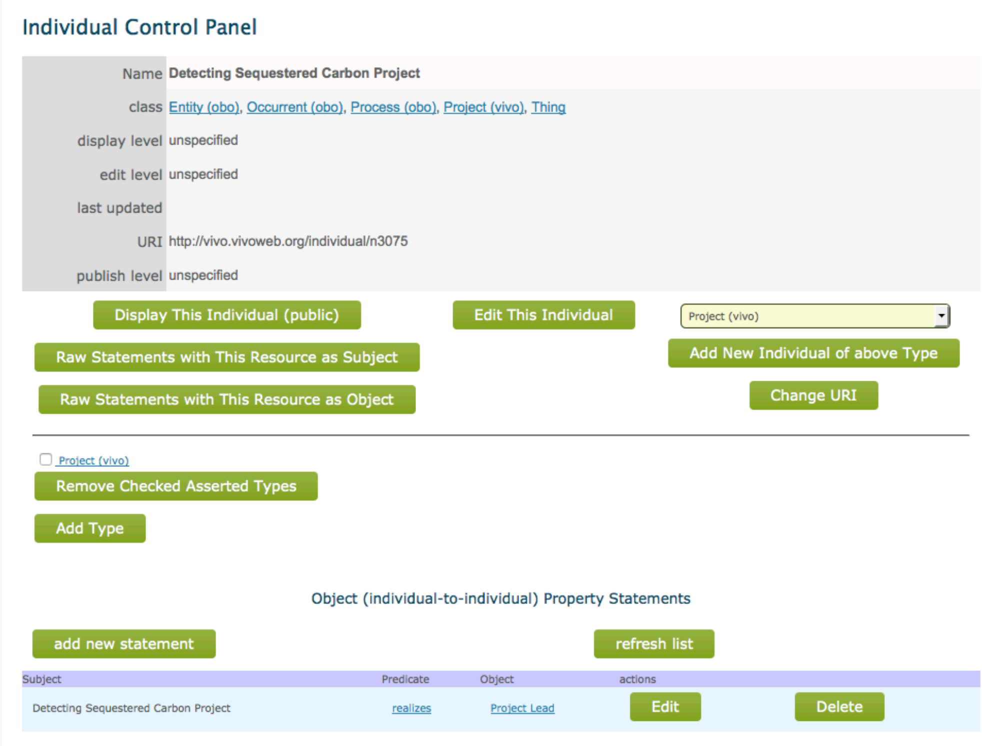
Display This Individual (public)
Edit This Individual
Researcher Role (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
☐ Researcher Role (vivo)
Remove Checked Asserted Types
Add Type

Object (individual-to-individual) Property Statements

add new statement
refresh list

Subject	Predicate	Object	actions	
Project Lead	realized in	Detecting Sequestered Carbon Project	Edit	Delete
Project Lead	inheres in	Curie, Marie T.	Edit	Delete
Project Lead	date/time interval	n3357	Edit	Delete

Finally, from the Project perspective, only the return (inverse) relationship to the Role is seen: the Project "realizes" (http://purl.obolibrary.org/obo/BFO_0000055) the Role.



Individual Control Panel

Name	Detecting Sequestered Carbon Project
class	Entity (obo), Occurrent (obo), Process (obo), Project (vivo), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n3075
publish level	unspecified

Display This Individual (public)
Edit This Individual
Project (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
☐ Project (vivo)
Remove Checked Asserted Types
Add Type

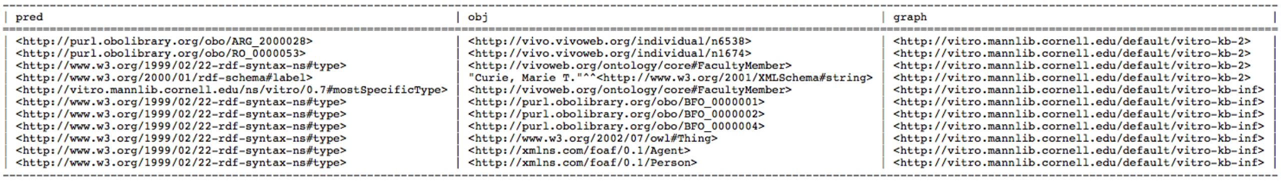
Object (individual-to-individual) Property Statements

add new statement
refresh list

Subject	Predicate	Object	actions	
Detecting Sequestered Carbon Project	realizes	Project Lead	Edit	Delete

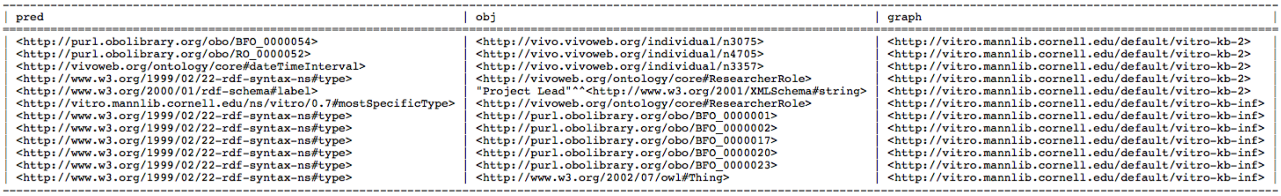
The triples underneath

For the Person (http://vivo.vivoweb.org/individual/n4705):



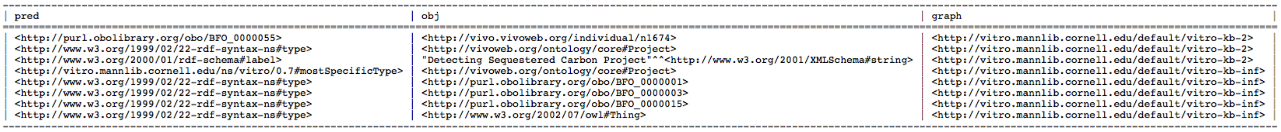
```
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                             | obj                                                          | graph                                                   |
=============================================================================================================================================================================================
| <http://purl.obolibrary.org/obo/ARG_2000028>                     | <http://vivo.vivoweb.org/individual/n6538>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://purl.obolibrary.org/obo/RO_0000053>                      | <http://vivo.vivoweb.org/individual/n1674>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://vivoweb.org/ontology/core#FacultyMember>             | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                     | "Curie, Marie T."^^<http://www.w3.org/2001/XMLSchema#string> | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#FacultyMember>             | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000001>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000002>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000004>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://www.w3.org/2002/07/owl#Thing>                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://xmlns.com/foaf/0.1/Agent>                            | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://xmlns.com/foaf/0.1/Person>                           | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

For the Role (http://vivo.vivoweb.org/individual/n1674):



```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                             | obj                                                       | graph                                                   |
==========================================================================================================================================================================================
| <http://purl.obolibrary.org/obo/BFO_0000054>                     | <http://vivo.vivoweb.org/individual/n3075>                | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://purl.obolibrary.org/obo/RO_0000052>                      | <http://vivo.vivoweb.org/individual/n4705>                | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vivoweb.org/ontology/core#dateTimeInterval>              | <http://vivo.vivoweb.org/individual/n3357>                | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://vivoweb.org/ontology/core#ResearcherRole>         | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                     | "Project Lead"^^<http://www.w3.org/2001/XMLSchema#string> | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#ResearcherRole>         | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000001>              | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000002>              | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000017>              | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000020>              | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000023>              | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://www.w3.org/2002/07/owl#Thing>                     | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

And finally, for the Project (http://vivo.vivoweb.org/individual/n3075):



```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                             | obj                                                                               | graph                                                   |
==================================================================================================================================================================================================================
| <http://purl.obolibrary.org/obo/BFO_0000055>                     | <http://vivo.vivoweb.org/individual/n1674>                                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://vivoweb.org/ontology/core#Project>                                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                     | "Detecting Sequestered Carbon Project"^^<http://www.w3.org/2001/XMLSchema#string> | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#Project>                                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000001>                                      | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000003>                                      | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://purl.obolibrary.org/obo/BFO_0000015>                                      | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>                | <http://www.w3.org/2002/07/owl#Thing>                                             | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

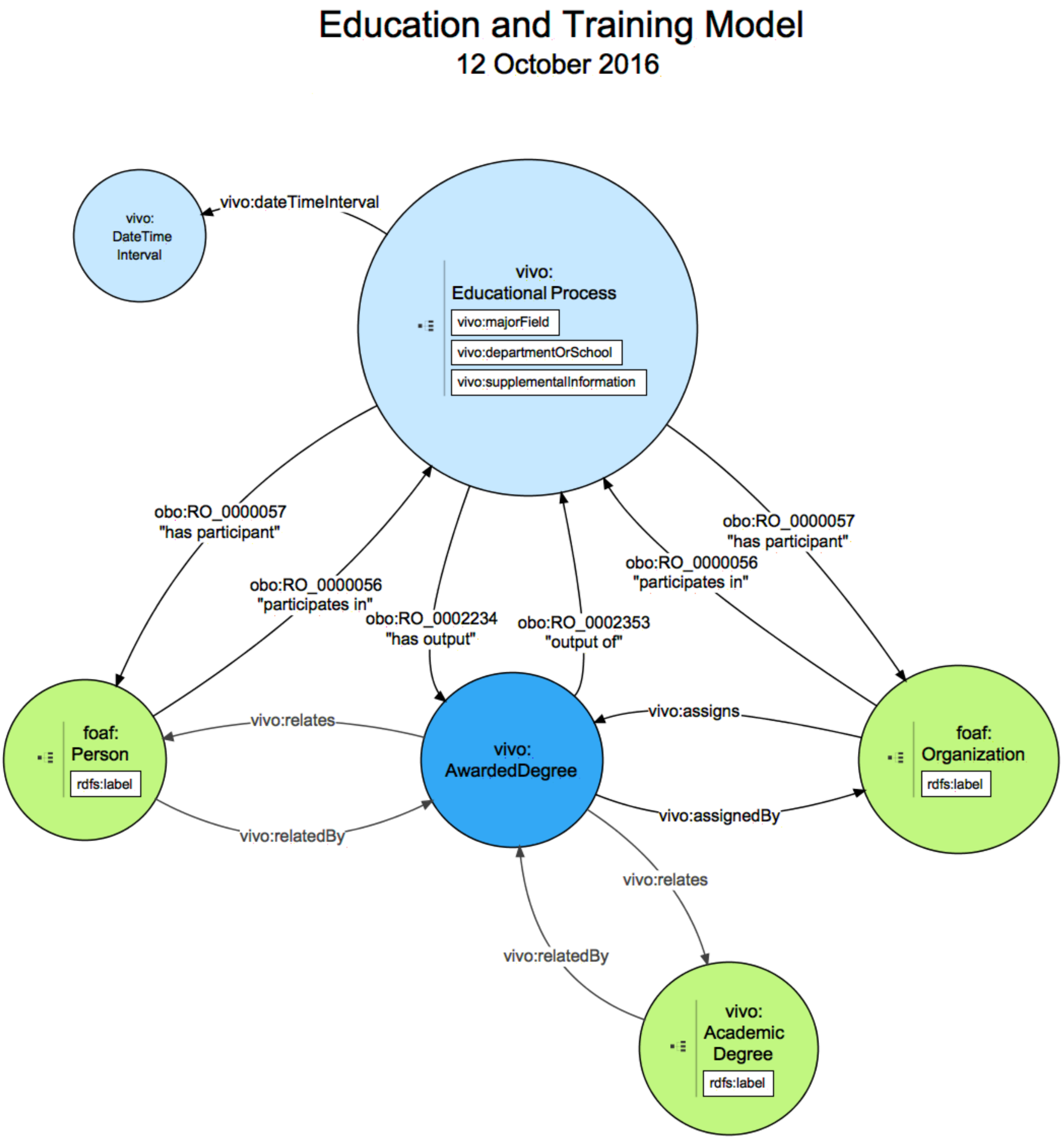
10.9.8.9 Education and Training Model

10.9.8.9.1 Notes

1. The entity of interest here is the AwardedDegree (dark blue in the center of the figure). The AwardedDegree is a relationship between a Person and an AcademicDegree. The AcademicDegree can be considered "abstract." The AwardedDegree is concrete – a person received the degree from a university at a particular time. VIVO provides a controlled vocabulary of AcademicDegrees. Note that the label for the degree is on the AcademicDegree.
2. The AwardedDegree has an associated EducationalProcess, which contains attributes of the AwardedDegree. The EducationalProcess has a DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 454) for detail.
3. See Organization Model (see page 451) for details regarding the modeling of organizations
4. For a list of AcademicDegrees, use the SPARQL query below

```
SELECT ?s ?name
WHERE {
    ?s a vivo:AcademicDegree .
    ?s rdfs:label ?name .
}
ORDER BY ?name
```

5. See Person Model (see page 458) for details regarding the modeling of people


Education and Training Model
12 October 2016
vivo: DateTime Interval
vivo:dateTimeInterval
vivo: Educational Process
vivo:majorField
vivo:departmentOrSchool
vivo:supplementalInformation
obo:RO_0000057 "has participant"
obo:RO_0000056 "participates in"
obo:RO_0002234 "has output"
obo:RO_0002353 "output of"
obo:RO_0000056 "participates in"
obo:RO_0000057 "has participant"
foaf: Person
rdfs:label
vivo:relates
vivo:relatedBy
vivo: AwardedDegree
vivo:assigns
vivo:assignedBy
foaf: Organization
rdfs:label
vivo:relates
vivo:relatedBy
vivo: Academic Degree
rdfs:label

Ontology Diagram Legend

Dark blue – the entity being modeled

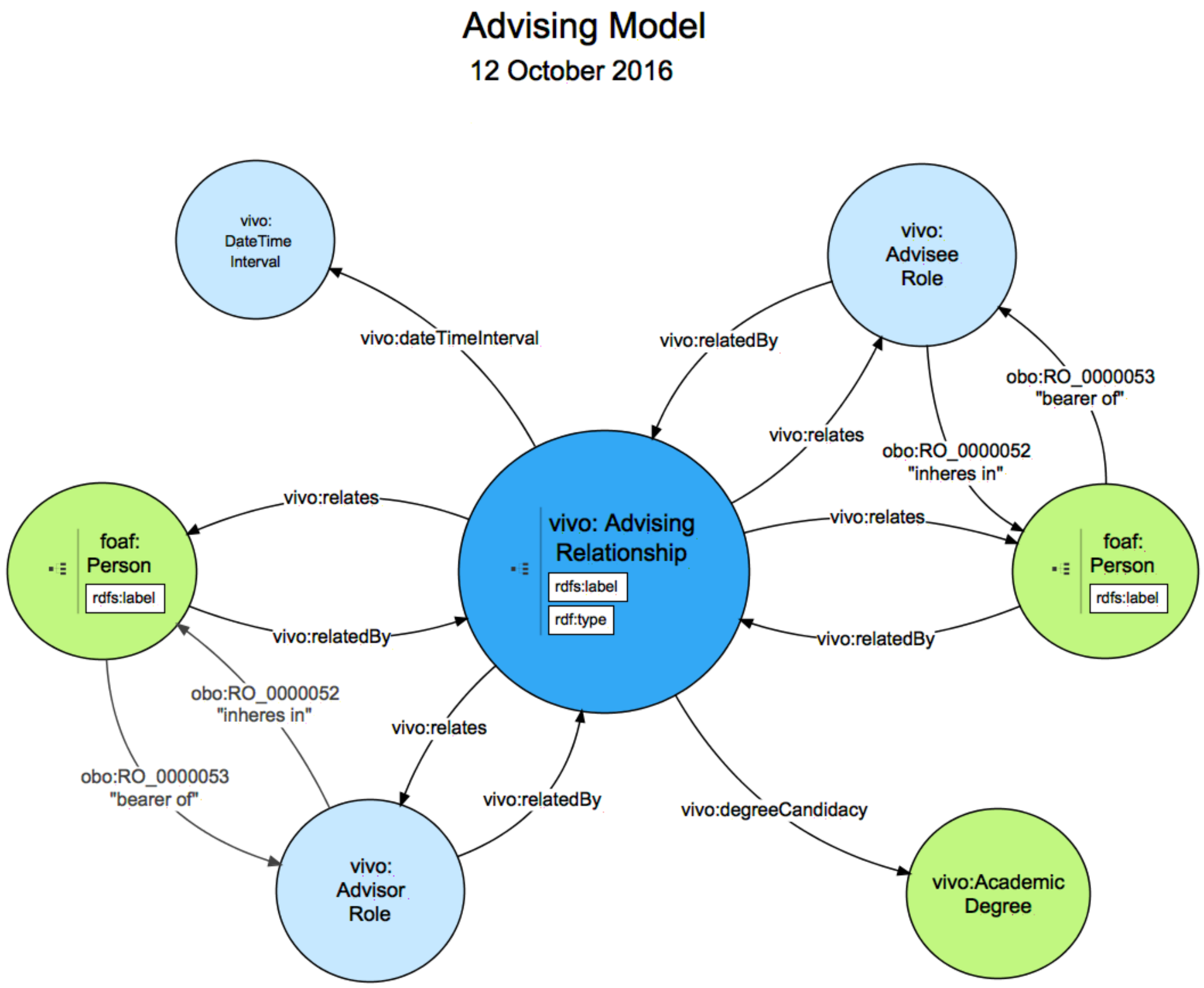
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.10 Advising Model

10.9.8.10.1 Notes

1. The label on the AdvisingRelationship is optional. VIVO constructs a label consisting of the Advisors label, the Advisee's label, some text representing the type of relationship
2. The AdvisingRelationship can optionally have one of the types below:
   ```
   http://vivoweb.org/ontology/core#FacultyMentoringRelationship
   http://vivoweb.org/ontology/core#GaduateAdvisingRelationship
   http://vivoweb.org/ontology/core#UndergraduateAdvisingRelationship
   http://vivoweb.org/ontology/core#PostdocOrFellowAdvisingRelationship
   ```
3. The AdviseeRole relates the AdvisingRelationship to the Advisee. This pattern is common for modeling roles and relationships.
4. The AcademicDegree is optional. It may be present for AdvisingRelationships leading to a degree.
5. The AdvisorRole relates the AdvisingRelationship to the Advisor. It uses the same pattern as the AdviseeRole.
6. The AdvisingRelationship may have an associated DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 454) for details regarding modeling DateTimeIntervals.


Advising Model
12 October 2016
vivo:
DateTime
Interval
vivo:
Advisee
Role
vivo:dateTimeInterval
vivo:relatedBy
obo:RO_0000053
"bearer of"
vivo:relates
obo:RO_0000052
"inheres in"
vivo:relates
foaf:
Person
rdfs:label
vivo: Advising
Relationship
rdfs:label
rdf:type
vivo:relates
foaf:
Person
rdfs:label
vivo:relatedBy
vivo:relatedBy
obo:RO_0000052
"inheres in"
vivo:relates
obo:RO_0000053
"bearer of"
vivo:relatedBy
vivo:degreeCandidacy
vivo:
Advisor
Role
vivo:Academic
Degree

Ontology Diagram Legend

Dark blue – the entity being modeled

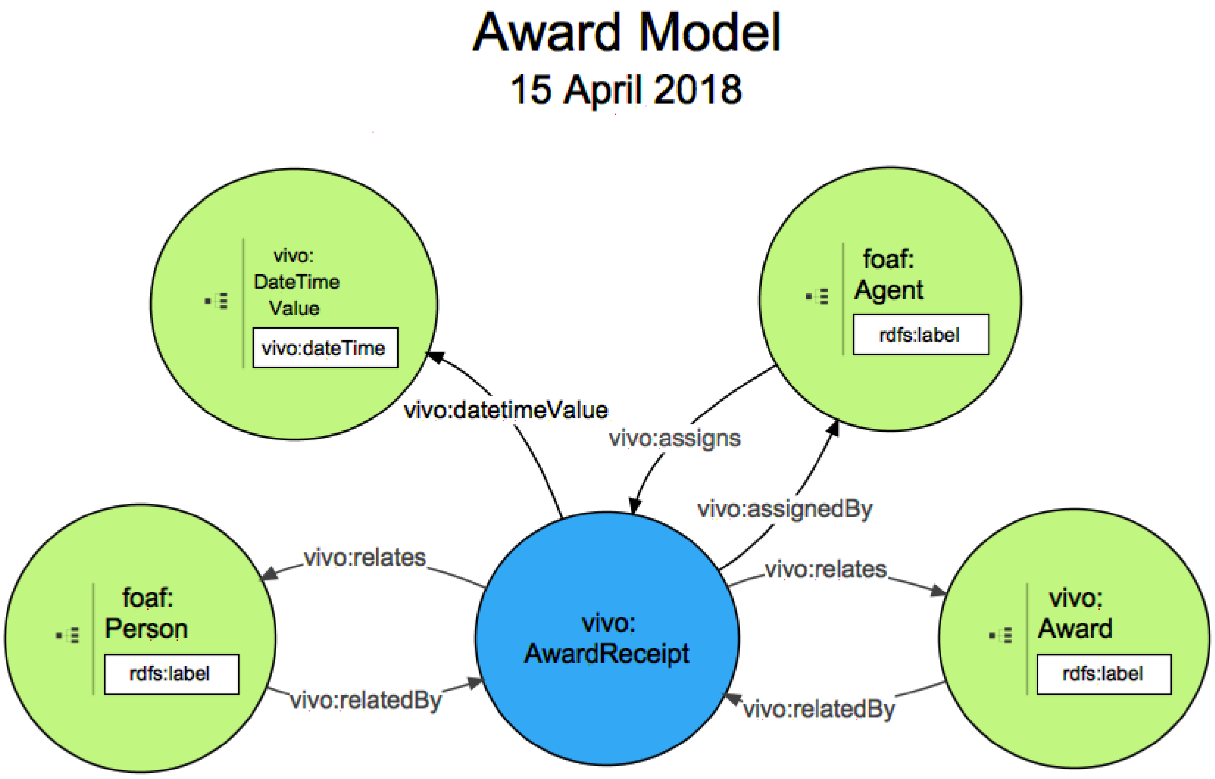
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.11 Award Model

10.9.8.11.1 Notes

1. The entity of interest here is the AwardReceipt. It is a relationship between a Person and an Award. The Award entity is generic, as in "The Nobel Prize in Physics." The AwardReceipt is specific, as in "Person x was awarded The Nobel Prize in Physics by the Royal Swedish Academy of Sciences on 10 October 2016"
2. The foaf:Agent is the entity making the award.


Award Model
15 April 2018
vivo:
DateTime
Value
vivo:dateTime
foaf:
Agent
rdfs:label
vivo:datetimeValue
vivo:assigns
vivo:assignedBy
foaf:
Person
rdfs:label
vivo:relates
vivo:relatedBy
vivo:
AwardReceipt
vivo:relates
vivo:relatedBy
vivo:
Award
rdfs:label

Ontology Diagram Legend

Dark blue – the entity being modeled

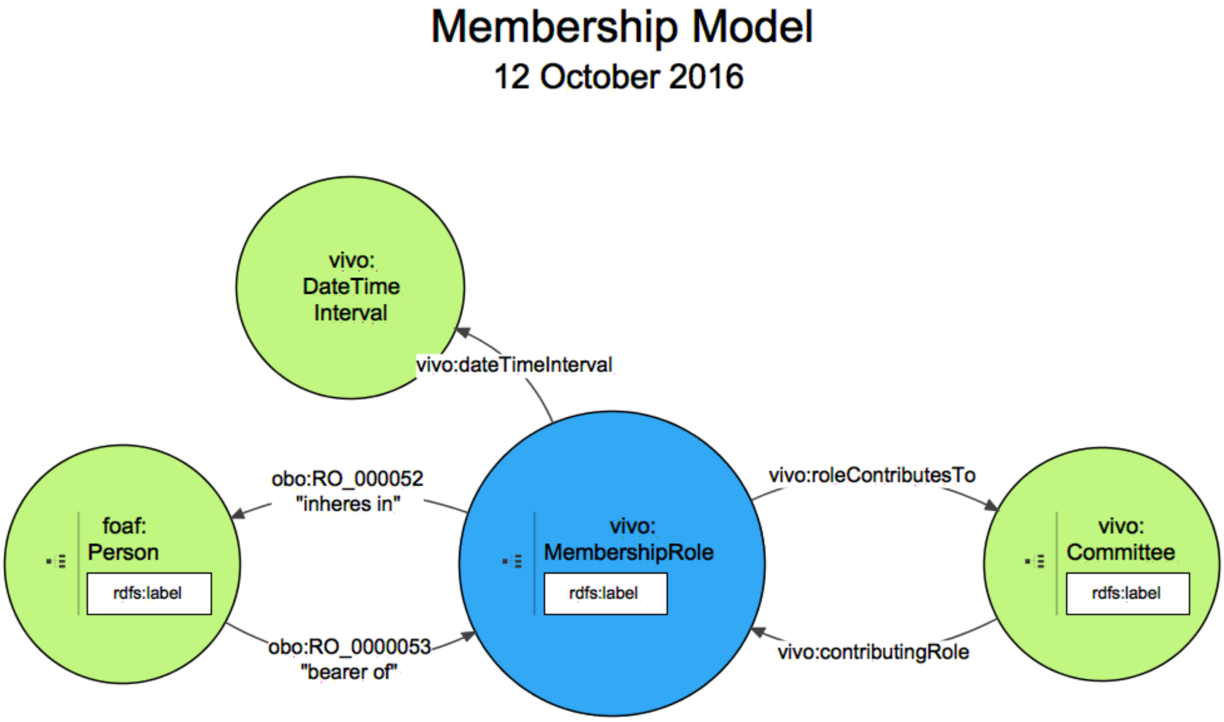
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.12 Membership Model

10.9.8.12.1 Notes

1. Membership is represented by using a MembershipRole to associate a person with an organization (a committee, or other organization).
2. The MembershipRole is associated with the organization using vivo:roleContributesTo and vivo:contributingRole
3. The MembershipRole has an optional label. The label is used to indicate the whether the person is "Chair" or "Member" or some other term that further describes the membership.


Membership Model
12 October 2016
vivo: DateTime Interval
vivo:dateTimeInterval
foaf: Person
rdfs:label
obo:RO_000052 "inheres in"
obo:RO_0000053 "bearer of"
vivo: MembershipRole
rdfs:label
vivo:roleContributesTo
vivo:contributingRole
vivo: Committee
rdfs:label

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.13 Ontology Diagram Legend

Dark blue – the entity being modeled

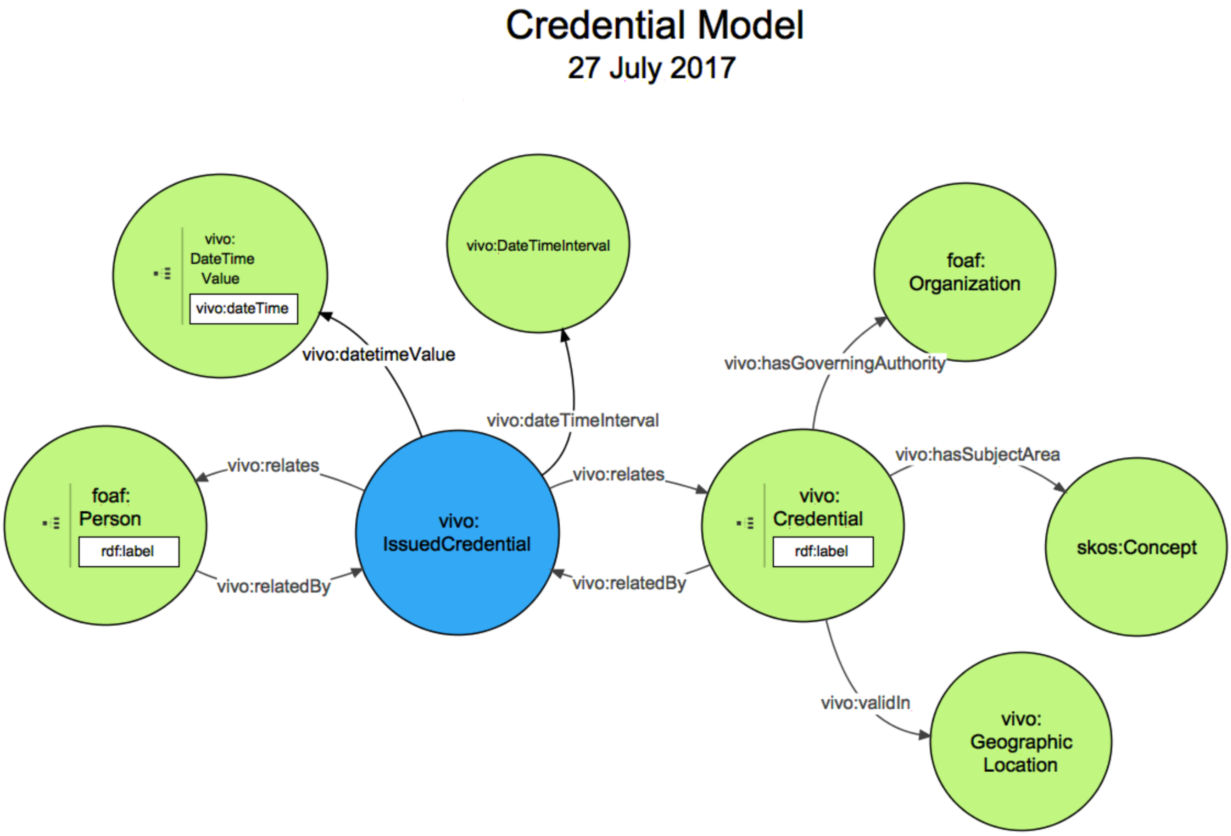
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.9.8.14 Credential Model

10.9.8.14.1 Notes

1. The entity of interest here is the IssuedCredential. It is a relationship between a Person and a Credential. The Credential entity is generic, as in "Board Certified in Neurology." The IssuedCredential is specific, as in "Person x was credentialed as Board Certified in Neurology in 2017"
2. The Credential may have a type of License or Certificate.
3. This model is similar to the Award Model.


Credential Model
27 July 2017
vivo:
DateTime
Value
vivo:dateTime
vivo:DateTimeInterval
foaf:
Organization
vivo:datetimeValue
vivo:hasGoverningAuthority
vivo:dateTimeInterval
vivo:relates
vivo:relates
vivo:hasSubjectArea
foaf:
Person
rdf:label
vivo:
IssuedCredential
vivo:
Credential
rdf:label
skos:Concept
vivo:relatedBy
vivo:relatedBy
vivo:validIn
vivo:
Geographic
Location

Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.10 Resource Links

The resources below should be helpful for anyone seeking additional information on topics related to VIVO, Vitro, ontologies, and the Semantic Web.

VIVO website	http://vivoweb.org/

VIVO project Wiki	https://wiki.duraspace.org/display/VIVO
VIVO project Facebook page	http://www.facebook.com/VIVOcollaboration
VIVO project Twitter	http://twitter.com/vivocollab
VIVO project LinkedIn group	https://www.linkedin.com/groups/2905369
Semantic Web technologies and Standards published by W3C	http://www.w3.org/2001/sw/wiki/Main_Page
Resource Description Framework is a standard model for data interchange on the web, to learn more about RDF	http://www.w3.org/2001/sw/wiki/RDF
More information on Web Ontology Language (OWL)	http://www.w3.org/2001/sw/wiki/OWL
SPARQL Query Language for RDF	http://www.w3.org/2001/sw/wiki/SPARQL
References to events, news, personal pages on the community	Semanticweb.org[183]
Semantic Web related conferences	the Semantic web “dogfood”[184]
Supporting the OWLED Workshop series and task forces	http://webont.org/owled/
Semantic Web portal dedicated to ontology design patterns (ODPs)	Ontology Design Pattern Wiki[185]
Books available on Semantic web development	http://www.w3.org/2001/sw/wiki/Books#Books_on_Semantic_Web:_Intro
Protégé is a lightweight web-based ontology editor supporting OWL	http://protege.stanford.edu

---

[183] http://semanticweb.org/
[184] http://data.semanticweb.org/
[185] http://ontologydesignpatterns.org/wiki/

10.11 Rich export SPARQL queries

VIVO's rich export queries are organized by typical sections of a curriculum vitae or CV.

They may be useful to supplement the examples of SPARQL queries[186] in the SPARQL Resources[187] section of this wiki.

Queries are grouped by directory in the /productMods/WEB-INF/rich-export section of the VIVO source code.

10.11.1 Rich export SPARQL queries: Address

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

address.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>


CONSTRUCT {
     ?address ?property ?object .
} WHERE {
     PERSON_URI obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasAddress ?address .
     ?address ?property ?object .
}
```

locationOfAddress.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

CONSTRUCT {
     ?geographicLocation ?property ?object .
} WHERE {
     PERSON_URI obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasAddress ?address .
     ?address obo:RO_0001025 ?geographicLocation .
     ?geographicLocation ?property ?object .
```

---

[186] https://wiki.lyrasis.org/display/VIVO/SPARQL+Queries+Regarding+Positions
[187] https://wiki.lyrasis.org/display/VIVO/SPARQL+Resources

```
}
```

10.11.2 Rich export SPARQL queries: Advising

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

advisee.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?advisee ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee ?property ?object .
}
```

adviseesDegreeAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?degree ?property ?object
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0002234 ?awardedDegree .
     ?awardedDegree core:relates ?degree .
     ?degree a core:AcademicDegree .
     ?degree ?property ?object
}
```

adviseesEducationalInstitutionAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?educationalInstitution rdfs:label ?label
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?educationalInstitution .
     ?educationalInstitution a foaf:Organization .
     ?educationalInstitution rdfs:label ?label
}
```

adviseesEducationalEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?dateTimeValue ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?dateTimeValue .
     ?dateTimeValue ?property ?object .
}
```

associatedDegree.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?degree ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:degreeCandidacy ?degree .
     ?degree ?property ?object .
}
```

associatedEducationalTraining.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:advisingContributionTo ?educationalTraining .
     ?educationalTraining ?property ?object .
}
```

associatedEducationalTrainingAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining ?property ?object .
}
```

10.11.3 Rich export SPARQL queries: Award

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

award.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?award ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:relates ?award .
     ?award a core:Award .
     ?award ?property ?object .
}
```

conferringOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:assignedBy ?organization .
     ?organization a foaf:Organization .
     ?organization ?property ?object .
}
```

sponsoringOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:relates ?award .
     ?award a core:Award .
     ?award core:sponsoredBy ?organization .
     ?organization ?property ?object .
}
```

10.11.4 Rich export SPARQL queries: Credential

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

credential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?credential ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:relates ?credential .
     ?credential a core:Credential .
     ?credential ?property ?object .
}
```

credentialGoverningAuthority.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:relates ?credential .
     ?credential a core:Credential .
     ?credential core:hasGoverningAuthority ?organization .
     ?organization ?property ?object .
}
```

elibibleForCredential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?credential ?property ?object .
} WHERE {
     PERSON_URI core:eligibleFor ?credential .
     ?credential ?property ?object .
}
```

issuedCredential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?issuedCredential ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential ?property ?object .
}
```

issuedCredentialExpirationDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:expirationDate ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

issuedCredentialIssueDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:dateIssued ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

issuedCredentialSubjectArea.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?subjectArea ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:hasSubjectArea ?subjectArea .
     ?subjectArea ?property ?object .
}
```

10.11.5 Rich export SPARQL queries: Educational Training

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

educationalTraining.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining ?property ?object .
}
```

educationalTrainingDegree.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?degree ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0002234 ?awardedDegree .
     ?awardedDegree a core:AwardedDegree .
     ?awardedDegree core:relates ?degree .
     ?degree a core:AcademicDegree .
     ?degree ?property ?object .
}
```

educationalTrainingEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?dateTimeInterval core:end ?date .
     ?date ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?date .
     ?date ?property ?object .
}
```

educationalTrainingLocation.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?geographicLocation ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?organization .
     ?organization a foaf:Organization .
     ?organization obo:RO_0001025 ?geographicLocation .
     ?geographicLocation a core:GeographicLocation .
     ?geographicLocation ?property ?object .
}
```

educationalTrainingOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?organization .
     ?organization a foaf:Organization .
     ?organization ?property ?object .
}
```

educationalTrainingStartDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?date ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:start ?date .
     ?date ?property ?object .
}
```

10.11.6 Rich export SPARQL queries: Funding

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

grantAwardedBy.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?awardingOrganization rdfs:label ?label
} WHERE {
    {
      {PERSON_URI core:relatedBy ?investigatorRole .
       ?investigatorRole a core:PrincipalInvestigatorRole
      }
        union
      {PERSON_URI core:relatedBy ?investigatorRole .
       ?investigatorRole a core:CoPrincipalInvestigatorRole
      }
     }

     ?investigatorRole core:relatedBy ?grant .
     ?grant a core:Grant .
     ?grant core:assignedBy ?awardingOrganization .
     ?awardingOrganization a foaf:Organization .
     ?awardingOrganization rdfs:label ?label
}
```

grants.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
     ?grant ?property ?object .
     ?investigatorRole core:relatedBy ?grant .
} WHERE {
     {
       { PERSON_URI core:relatedBy ?investigatorRole .
         ?investigatorRole a core:PrincipalInvestigatorRole
       }
         union
       { PERSON_URI core:relatedBy ?investigatorRole .
         ?investigatorRole a core:CoPrincipalInvestigatorRole
       }
     }

     ?investigatorRole core:relatedBy ?grant .
     ?grant a core:Grant .
     ?grant ?property ?object
}
```

10.11.7 Rich export SPARQL queries: Membership

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

memberRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?endeavor ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?memberRole .
     ?memberRole a core:MemberRole .
     ?memberRole core:roleContributesTo ?endeavor .
     ?endeavor ?property ?object .
}
```

10.11.8 Rich export SPARQL queries: Outreach

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

outreachRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?endeavor ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?outreachRole .
     ?outreachRole a core:OutreachProviderRole .
     ?outreachRole core:roleContributesTo ?endeavor .
     ?endeavor ?property ?object .
}
```

10.11.9 Rich export SPARQL queries: Patent

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

assignee.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?assignee ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:assignee ?assignee .
     ?assignee ?property ?object .
}
```

inventors.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?person ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?authorship core:relates ?person .
     ?person a foaf:Person .
     ?person ?property ?object .
}
```

patent.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?patent ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent ?property ?object .
}
```

patentFiledDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:dateFiled ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

patentIssuedDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:dateIssued ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

10.11.10 Rich export SPARQL queries: Position

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

locationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?organization core:hasGeographicLocation ?geographicLocation .
     ?geographicLocation rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:RO_0001025 ?geographicLocation .
     ?geographicLocation rdfs:label ?label .
}
```

organizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?position core:positionInOrganization ?organization .
     ?organization rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization rdfs:label ?label .
}
```

subOrganizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?organization core:hasSubOrganization ?subOrganization .
     ?subOrganization rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:BFO_0000050 ?subOrganization .
     ?subOrganization rdfs:label ?label .
}
```

superOrganizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?superOrganization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:BFO_0000051 ?superOrganization .
     ?superOrganization ?property ?object .
}
```

10.11.11 Rich export SPARQL queries: Presentation

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

meetingLocation.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?location rdfs:label ?locationName .
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation obo:BFO_0000050 ?containingEvent .
     ?containingEvent obo:RO_0001025 ?location .
     ?location rdfs:label ?locationName .
}
```

meetingName.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?containingEvent rdfs:label ?containingEventName
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation obo:BFO_0000050 ?containingEvent .
     ?containingEvent rdfs:label ?containingEventName
}
```

presenterRoleIn.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?presentation rdfs:label ?presentationTitle .
     ?presenterRole rdfs:label ?roleLabel .
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole rdfs:label ?roleLAbel .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation rdfs:label ?presentationTitle .
}
```

10.11.12 Rich export SPARQL queries: Publication

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

associatedJournal.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publicationVenue ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:hasPublicationVenue ?publicationVenue .
     ?publicationVenue ?property ?object .
}
```

authors.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?coAuthorship ?property1 ?object1 .
     ?person ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:relatedBy ?coAuthorship .
     ?coAuthorship a core:Authorship .
     ?coAuthorship ?property1 ?object1 .
     ?coAuthorship core:relates ?person .
     ?person a foaf:Person .
     ?person ?property2 ?object2 .
}
```

presentedAtEvent.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?event ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
}
```

presentedAtEventEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>PREFIX bibo: <http://purl.org/ontology/bibo/
>

CONSTRUCT {
     ?endDate ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
     ?event core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?endDate .
     ?endDate core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

presentedAtEventLocation.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?location rdfs:label ?locationName .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event obo:RO_0001025 ?location .
     ?location rdfs:label ?locationName .
}
```

presentedAtEventStartDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?startDate ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
     ?event core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:start ?startDate .
     ?startDate core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

publication.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publication ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication ?property ?object .
}
```

publicationDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication ?dateTimeValue ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

publicationPartOfInfoResource.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?informationResource ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication obo:BFO_0000050 ?informationResource .
     ?informationResource ?property ?object .
}
```

publicationReproducedIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?informationResource ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:reproducedIn ?informationResource .
     ?informationResource ?property ?object .
}
```

publicationStatus.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publicationStatus ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:status ?publicationStatus .
     ?publicationStatus ?property ?object .
}
```

publicationURL.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

CONSTRUCT {
     ?urllink ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasURL ?urllink .
     ?urllink ?property ?object .
}
```

publisher_variant1.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publisher ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:hasPublicationVenue ?publicationVenue .
     ?publicationVenue core:publisher ?publisher .
     ?publisher ?property ?object .
}
```

publisher_variant2.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publisher ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:publisher ?publisher .
     ?publisher ?property ?object .
}
```

10.11.13 Rich export SPARQL queries: Teaching

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

teacherRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?course ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?teacherRole .
     ?teacherRole a core:TeacherRole .
     ?teacherRole obo:BFO_0000054  ?course .
     ?course ?property ?object .
}
```

10.12 URL Reference

10.12.1 Overview

VIVO has several pages that can be reached by adding one of the words below to the end of your VIVO URL. For example, if the URL of your VIVO home page is `http://vivo.myschool.edu`, then you can access a page regarding revision information by accessing `http://vivo.myschool.edu/revisionInfo`.

10.12.2 `api/datarequest`

The Data Distribution API (see page 339) endpoint, typically followed by an action name of a configured data distributor.

10.12.3 `freemarkersamples`

Displays a page of Freemarker widget results. The template for this page can be found here vitro-core/webapp/web/templates/freemarker/body/samples.ftl.

10.12.4 `login`

Takes you to the VIVO login page if you ar enot logged in, or a message indiating that you are logged in.

10.12.5 `logout`

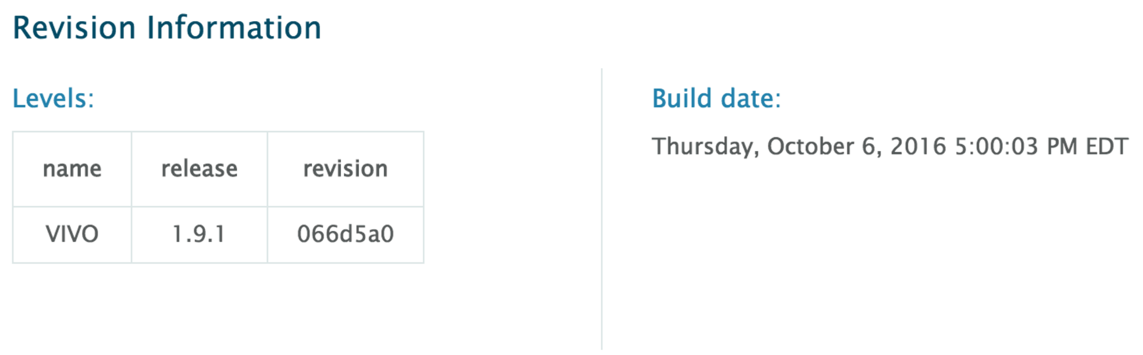
Logs you out of VIVO.

10.12.6 `RecomputeInferences`

Review all triples in the triple store and add inferences to the inference graph as needed.

10.12.7 `revisionInfo`

Show a page of version information including date and time of most recent build, as shown below.



Revision Information

Levels:

name	release	revision
VIVO	1.9.1	066d5a0

Build date:

Thursday, October 6, 2016 5:00:03 PM EDT

10.12.8 `sitemap.xml`

See the XML VIVO generates for your site's sitemap.

10.12.9 `SiteAdmin`

Shows the Site Admin page.

10.12.10 `SearchIndex`

Show search index status and access to rebuild the VIVO search index

10.12.11 `tpf/core`

The Triple Pattern Fragments (see page 380) endpoint

10.12.12 `vivosolr`

Display the VIVO Solr search index control panel

10.13 Utilities Reference

- jena2tools (see page 503)

- jena3tools (see page 503)

10.13.1 jena2tools

jena2tools is a command line Java utility which uses the Jena libraries to export data from a Jena 2 VIVO system (versions 1.9.3 and earlier). Jena2tools exports both the data store and the configuration store. The exports are stored in `vivo-home/dumps`.

jena2tools is used during an upgrade process from VIVO versions running Jena 2 (version 1.9.3 and earlier). See Upgrading VIVO (see page 46)

jena2tools can also be used to export your VIVO configuration and content for sharing, and/ or use in other tools. See Exporting Data from VIVO (see page 109)

10.13.1.1 Running Jena2tools

1. Shutdown Tomcat
2. Use `java -jar jena2tools.jar` <options>
3. Your dumps will be in `vivo-home/dumps`

10.13.1.2 Help

```
java -jar jena2tools [arguments] -d home_directory

Arguments
-d, --dir      REQUIRED. Specify the VIVO/Vitro home directory
-i, --import   Import TriG format data to triple stores
-e, --export   Export data from triple stores. Default format is TriG
-h, --help     Display help text
-f, --force    Force overwrite of previous exports
-o, --output   Output format followed by one of nt, nq, jsonld, trig, rdf, or ttl
```

10.13.1.3 Repository

Source code for Jena2tools can be found in GitHub here: https://github.com/vivo-project/jenatools

10.13.2 jena3tools

jena3tools is a command line Java utility which uses the Jena libraries to import and export data to and from a Jena 3 VIVO system (version 1.10 and beyond). Jena3tools imports and exports both the data store and the configuration store. The exports are stored in `vivo-home/dumps`.

jena3tools is used during an upgrade process from VIVO versions running Jena 2 (version 1.9.3 and earlier). See Upgrading VIVO (see page 46)

jena3tools can also be used to export your VIVO configuration and content for sharing, and/ or use in other tools. See Exporting Data from VIVO (see page 109)

10.13.2.1 Running Jena2tools

1. Shutdown Tomcat
2. Use `java -jar jena3tools.jar` <options>
3. Your dumps will be in `vivo-home/dumps`

10.13.2.2 Help

```
java -jar jena3tools [arguments] -d home_directory

Arguments
-d, --dir      REQUIRED. Specify the VIVO/Vitro home directory
-i, --import   Import TriG format data to triple stores
-e, --export   Export data from triple stores. Default format is TriG
-h, --help     Display help text
-f, --force    Force overwrite of previous exports
-o, --output   Output format followed by one of nt, nq, jsonld, trig, rdf, or ttl
```

10.13.2.3 Repository

Source code for Jena3tools can be found in GitHub here: https://github.com/vivo-project/jenatools

11 About This Documentation

VIVO documentation is created by the users of VIVO.

If you find something here that is incorrect or confusing or incomplete, please let us know by posting on vivo-tech@googlegroups.com[188]

If you would like to write, rewrite or otherwise improve this documentation, please contact @Dragan Ivanovic

11.1 Maintaining release-specific info on the Wiki

11.1.1 Goals

Recognize that some documentation applies only to a particular release, or set of releases.
Recognize that documentation will most likely be found by web searches, not by walking the wiki.
Information about the current release should be the easiest to find.
Information about older releases should be available somewhere.
There should be an area for documenting new features before they are included in a release.
It should be easy to tell whether the information you are viewing is correct for the code you are using.
Documentation should not be frozen at release time - It should remain available for improvements.
The most basic instructions should be included in the release.
Novice users should be able to find what they need; expert users should be able to find more

11.1.2 Two types of wiki pages

11.1.2.1 Release-specific pages

- Apply to a specific release, or range of releases.
- Incorrect if used with an inappropriate release.
- Examples: installation instructions, customization guides, ontology details

11.1.2.2 Release-neutral pages

- Equally relevant to all releases (or to none)
- Examples: tutorials, meeting agendas, glossary, outreach events and resources, presentations.
- Most wiki pages are not release specific

---

[188] mailto:vivo-tech@googlegroups.com

11.1.3 Approach

11.1.3.1 VIVO (main wiki, also known as the project wiki, also known as the community wiki)

- Located at http://wiki.duraspace.org/display/VIVO
- The main wiki holds all sorts of information, as it does now.
- Includes governance, task forces, interest groups, background material, community support materials
- Does not include release specific information describing the product

11.1.3.2 VIVO Release specific wikis, also known as the documentation

- The release-specific information will be collected in release specific wikis. These wikis will be copied forward to create spaces for new releases. The next release wiki will be available before the release of the software to document the next release.
- Contains release-specific pages for older releases.
- The styling indicates that the wiki id documentation for a specific release
- Release specific wikis exist for releases prior to the current release, for the current release and for the next release
- The release specific wikis use a documentation template and a documentation PDF export template optimized for the production of a single PDF document from the documentation wiki.

11.1.3.3 Minimal documentation in the Git repository

- The release specific documentation wiki is the definitive documentation for the current release
- The project wiki is the definitive source of material regarding the project
- The Git README.md refers to the project wiki and the release specific wiki and describes each

11.1.4 Between releases

- All community processes continue in the project wiki
- Release specific wikis are available for improvement
- A next release wiki is created from the current release wiki when there are new features to document.

11.2 VIVO documentation style guide

Each VIVO document consists of a large number of wiki pages. These pages must work well together, both on the wiki and when exported to a PDF file. Here are some suggestions to help that happen.

11.2.1 Page sizes

Keep each page to a manageable size, and focused on a particular topic. If you find that the page is too complex or too diverse, break it into smaller pages. Within each group of pages, the parent should contain introductory material or an overview, while the child pages explore individual topics.

11.2.2 Start with a Table of Contents

Start the page with a call to the "Table of Contents" macro. The Table of Contents will include all of the headings in the current page. It will also include a top-level heading for each child page, thanks to the "Children Display" macro.

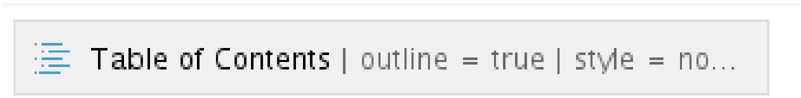
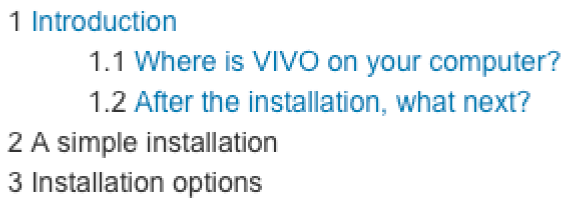


Table of Contents | outline = true | style = no...

3 Table of Contents (editing)



1 Introduction
    1.1 Where is VIVO on your computer?
    1.2 After the installation, what next?
2 A simple installation
3 Installation options

4 Table of Contents (display)

When the document is exported to a PDF file, the "Table of Contents" macros are not included. Instead, the file begins with a table of contents for the full document.

11.2.3 Use all heading levels

The major headings on all pages should be Heading 1. The second headings on all pages should be Heading 2. Use the Heading styles only – do not format headings using bold, italics, etc. When the pages are combined into a PDF file, the heading levels will be displayed properly and numbered correctly in the table of contents and in the document.

Pagination in the PDF document is controlled by the PDF template, not by the author. Use page titles and page headings consistently. The resulting PDF document will then be consistent.

Headings within the child pages are demoted accordingly, to keep the organization intact. So a level 2 heading in the Table of Contents might represent:

- a level 2 heading in the parent page,
- a level 1 heading in a child page,
- the title of a grandchild page.

11.2.4 Code

Use a code block macro to represent code. A title for the code block is optional.

Use monospace in sentences to represent code.

11.2.5 Linking within the document

Links in the technical documentation wiki need to be checked to make sure they are referring to other parts of the tech doc, or external sources. Links to pages in the project wiki are “okay” but need to be done carefully -- such links are often red flags that the tech doc is drifting into content better suited for the project wiki. Links to the archive should be avoided. Content from the archive that is relevant for the current version should be copied into the documentation wiki.

11.2.6 End with a Children Display macro

The documentation wiki for VIVO includes a child display at the end of every page. There is no need to include a children display macro explicitly. It will be added for you.