

CAGRID 1.2

Programmer's Guide



This is a U.S. Government work.

March 26, 2008



caBIG™

cancer Biomedical
Informatics Grid™

Model caBIG™ Open Source Software License

v.2

Release Date: January 7, 2008

Copyright Notice. Copyright 2008 The Ohio State University Research Foundation (OSURF), Argonne National Labs (ANL), SemanticBits LLC (SemanticBits), and Ekagra Software Technologies Ltd. (Ekagra) ("caBIG™ Participant"). The caGrid 1.2 software was created with NCI funding and is part of the caBIG™ initiative. The software subject to this notice and license includes both human readable source code form and machine readable, binary, object code form (the "caBIG™ Software").

This caBIG™ Software License (the "License") is between caBIG™ Participant and You. "You" (or "Your") shall mean a person or an entity, and all other entities that control, are controlled by, or are under common control with the entity. "Control" for purposes of this definition means (i) the direct or indirect power to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

License. Provided that You agree to the conditions described below, caBIG™ Participant grants You a non-exclusive, worldwide, perpetual, fully-paid-up, no-charge, irrevocable, transferable and royalty-free right and license in its rights in the caBIG™ Software, including any copyright or patent rights therein, to (i) use, install, disclose, access, operate, execute, reproduce, copy, modify, translate, market, publicly display, publicly perform, and prepare derivative works of the caBIG™ Software in any manner and for any purpose, and to have or permit others to do so; (ii) make, have made, use, practice, sell, and offer for sale, import, and/or otherwise dispose of caBIG™ Software (or portions thereof); (iii) distribute and have distributed to and by third parties the caBIG™ Software and any modifications and derivative works thereof; and (iv) sublicense the foregoing rights set out in (i), (ii) and (iii) to third parties, including the right to license such rights to further third parties. For sake of clarity, and not by way of limitation, caBIG™ Participant shall have no right of accounting or right of payment from You or Your sublicensees for the rights granted under this License. This License is granted at no charge to You. Your downloading, copying, modifying, displaying, distributing or use of caBIG™ Software constitutes acceptance of all of the terms and conditions of this Agreement. If you do not agree to such terms and conditions, you have no right to download, copy, modify, display, distribute or use the caBIG™ Software.

1. Your redistributions of the source code for the caBIG™ Software must retain the above copyright notice, this list of conditions and the disclaimer and limitation of liability of Article 6 below. Your redistributions in object code form must reproduce the above copyright notice, this list of conditions and the disclaimer of Article 6 in the documentation and/or other materials provided with the distribution, if any.
2. Your end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Ohio State University Research Foundation (OSURF), Argonne National Labs (ANL), SemanticBits LLC (SemanticBits), and Ekagra Software Technologies Ltd. (Ekagra)." If You do not include such end-user

documentation, You shall include this acknowledgment in the caBIG™ Software itself, wherever such third-party acknowledgments normally appear.

3. You may not use the names "The Ohio State University Research Foundation", "OSURF", "Argonne National Labs", "ANL", "SemanticBits LLC", "SemanticBits", "Ekagra Software Technologies Ltd.", "Ekagra", "The National Cancer Institute", "NCI", "Cancer Bioinformatics Grid" or "caBIG™" to endorse or promote products derived from this caBIG™ Software. This License does not authorize You to use any trademarks, service marks, trade names, logos or product names of either caBIG™ Participant, NCI or caBIG™, except as required to comply with the terms of this License.
4. For sake of clarity, and not by way of limitation, You may incorporate this caBIG™ Software into Your proprietary programs and into any third party proprietary programs. However, if You incorporate the caBIG™ Software into third party proprietary programs, You agree that You are solely responsible for obtaining any permission from such third parties required to incorporate the caBIG™ Software into such third party proprietary programs and for informing Your sublicensees, including without limitation Your end-users, of their obligation to secure any required permissions from such third parties before incorporating the caBIG™ Software into such third party proprietary software programs. In the event that You fail to obtain such permissions, You agree to indemnify caBIG™ Participant for any claims against caBIG™ Participant by such third parties, except to the extent prohibited by law, resulting from Your failure to obtain such permissions.
5. For sake of clarity, and not by way of limitation, You may add Your own copyright statement to Your modifications and to the derivative works, and You may provide additional or different license terms and conditions in Your sublicenses of modifications of the caBIG™ Software, or any derivative works of the caBIG™ Software as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
6. THIS caBIG™ SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES (INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE) ARE DISCLAIMED. IN NO EVENT SHALL THE OHIO STATE UNIVERSITY RESEARCH FOUNDATION ("OSURF"), ARGONNE NATIONAL LABS ("ANL"), SEMANTICBITS LLC ("SEMANTICBITS"), AND EKAGRA SOFTWARE TECHNOLOGIES LTD. (EKAGRA) OR ITS AFFILIATES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS caBIG™ SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision History

The following is the revision history for this document.

Date	Version	Description	Revised By
3/29/2008	1.2.2	Added note in WebSSO Section	Kunal Modi
3/23/2008	1.2.1	Formatted and Edited to Match NCI Documentation Standards	Carolyn Klinger
3/17/2008	1.2.0	Updated Release: caGrid 1.2	Scott Oster
9/20/2007	1.1	Updated Release: caGrid 1.1	Scott Oster
12/18/2006	1.0	Initial Release: caGrid 1.0	Scott Oster

Contents

Revision History	v
Contents	i
Figures.....	v
Tables	vii
About This Guide	1
Purpose	1
Release Schedule	1
Audience.....	1
Topics Covered	2
Document Text Conventions	2
Credits and Resources	3
Chapter 1 Overview of caGrid	5
Introduction.....	5
Standards Compliant.....	6
Model Driven	6
Semantically Discoverable	7
Secure and Manageable	9
Revolutionary Development	12
Chapter 2 caGrid Release Structure.....	15
Overview.....	15
caGrid Projects	15
Multiple Grid Support.....	23
Process	23
Issues	25
Depending on caGrid.....	25
Ivy Introduction.....	25
Depending on caGrid using Ivy	26
A simple example project appropriate for caGrid	26
Integrating Ivy into the project.....	27
Using Ivy to extract libraries from a local caGrid installation	28
Using Ivy to download libraries from the remote caGrid repository.....	34
Looking at Ivy reports as a guide to manually copying libraries from a local caGrid installation	38
Ivy Summary	39
Chapter 3 caGrid Metadata Infrastructure	41
Code Example Information	41
Metadata API Usage Overview	41
API Details	44
API Usage Examples	46
Discovery API Usage Overview	49
API Details	51
API Usage Examples	63
caDSR Grid Service Usage Overview	68
API Details	71
API Usage Examples	75
EVS API Usage Overview	79
getMetaSources	79
Examples of Use	80

getVocabularyNames.....	80
searchDescLogicConcept.....	80
getHistoryRecords	81
searchMetaThesaurus	82
searchSourceByCode	84
Chapter 4 caGrid Security	87
Dorian Overview	87
Creating a Grid Proxy Programmatically.....	89
Grid Grouper Overview	90
Grid Grouper Object Model.....	92
Credential Delegation Service (CDS) Overview	94
Programmatically Delegating a Credential (Identity Delegation Policy).....	95
Programmatically Delegating a Credential (Group Delegation Policy)	96
Programmatically Obtaining a Delegated Credential	98
WebSSO (Web Single Sign On) Overview.....	99
Introduction	99
Out of Box SSO Component.....	99
JA-SIG's Central Authentication Service (CAS).....	99
Overview of WebSSO Framework	100
Overview of WebSSO Workflow	100
Single Sign in Workflow	102
WebSSO Installation/ Configuration	103
WebSSO Server Installation.....	103
Web Single Sign On (WebSSO) Server Installation And Configuration.....	103
WebSSO Client Integration	108
Web Single Sign Out Client (WebSSO Client) Installation And Configuration.....	108
WebSSO Client APIs.....	118
Retrieving User Attributes Inside Client Application.....	118
Chapter 5 caGrid Service Developers.....	121
Introduce Service Developers	121
Deserialization and Serialization Examples	121
caGrid Debugging Information	121
Using the Client	123
Advanced Client Usage	124
Printing detailed faults.....	124
Using Subscription and Notification	124
Loading a proxy from a file.....	125
Implementing the Service.....	125
Advanced Service Development Topic	125
Obtaining the caller's identity	125
Accessing service properties	126
Getting a Handle to a Resource.....	126
Creating a Security Descriptor for a Resource	126
Setting the Termination Time on a Resource	128
Utilizing the Factory Pattern.....	129
Introduce Toolkit Developers.....	131
Creating an Update Site.....	131
Writing an Extension	132
Bundling an Extension	132
Chapter 6 caGrid Data Services	133
Overview	133
CQL	133
Creating a Query	135
Programmatic query building	135

Loading queries from files	136
Generic Data Service Clients	136
Client Side Utilities	137
Most Current Information and Examples.....	139
Utility Classes	139
Utilities.....	139
Validation Tools.....	141
CQL Query Processors	143
Overview	143
Implementation.....	143
Service Styles Architecture.....	146
Functionality Extended by Styles	146
Developing Data Service Styles.....	147
Style Library Directory.....	149
Federated Query Processor Usage Overview	149
Federated Query Engine.....	152
Federated Query Processor Service	152
Federated Query Results Service	153
Security Considerations	153
API Details.....	154
gov.nih.nci.cagrid.fqp.client.FederatedQueryProcessorClient	154
gov.nih.nci.cagrid.fqp.results.client.FederatedQueryResultsClient.....	155
gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine	157
API Usage Examples	158
Removing Query Results	160
Scheduling Removal of Query Results	161
Using the Engine Directly to Access Protected Data Services	162
Chapter 7 WS-Enumeration.....	164
Overview.....	164
Enumeration Service Context	164
Extension Implementation.....	164
Utilizing Enumeration in a Grid Service	164
Enumeration Implementations	165
Creating an Enumeration Response	165
Example	166
Client API.....	166
ClientEnumeration.....	166
Utilities.....	169
Examples	169
Basic caGrid WS-Enumeration	169
Using BDT	170
Chapter 8 Workflow Management Service	171
Overview.....	171
Workflow Architecture.....	171
WorkflowFactoryService API	173
Public WorkflowFactoryOutputType createWorkflow(WorkflowDescriptionType wmsInputType) throws WorkflowException ()	174
WorkflowServiceImpl API	175
Public WorkflowStatusType start(StartInputType input) throws WorkflowException, StartCalledOnStartedWorkflowFault	175
Security in WorkflowFactory and Context Services.....	178
Service Selection	178
Provenance Tracking.....	179
WS-RF Resources in Workflows	179
Variables	179

Creating a Web Service Instance	180
Invoking the Web Service Instance.....	180
Accessing resource properties.....	183
API Sample for WorkflowFactory and WorkflowImpl Services:.....	185
Chapter 9 caGrid Global Model Exchange	187
Overview	187
Building the GME	187
Configuring and Deploying the GME	188
GME Client	191
GME Client API.....	191
GME Viewer.....	191
Scientific Publications.....	193
Technical Manuals/Articles.....	196
caBIG Material.....	196
caCORE Material	197

Figures

Figure 1-1 caGrid Data Description Infrastructure	7
Figure 1-2 caGrid Discovery Overview	9
Figure 1-3 GAARDS Security Infrastructure	10
Figure 3-1 Service Model.....	42
Figure 3-2 Data Service metadata model	43
Figure 3-3 Metadata Exceptions	44
Figure 3-4 Accessing standard service metadata.....	47
Figure 3-5 Accessing standard data service metadata.....	47
Figure 3-6 Serializing metadata to a file	48
Figure 3-7 Deserializing metadata from a file	49
Figure 3-8 Metadata exceptions	50
Figure 3-9 Common DiscoveryClient exception	51
Figure 3-10 DiscoveryClient constructor example	64
Figure 3-11 Configuring Index Service code	65
Figure 3-12 Discovering all services	65
Figure 3-13 Discovering by Search String	66
Figure 3-14 Discovering Data Services by Model Name	66
Figure 3-15 Discovering by Research Center.....	66
Figure 3-16 Discovering Services by Concept Code	67
Figure 3-17 Discover Services by Point of Contact	67
Figure 3-18 Discover Services by Input.....	67
Figure 3-19 UML project model	70
Figure 3-20 caDSR Grid Service Types	71
Figure 3-21 CaDSRServiceClient Inheritance Graph	72
Figure 3-22 Navigating projects registered in the caDSR.....	76
Figure 3-23 Generating a Domain Model for a Whole Project.....	77
Figure 3-24 Generating a domain model for a package	78
Figure 3-25 Generating a restricted domain model	79
Figure 4-1 Dorian.....	89
Figure 4-2 Programmatically creating a grid proxy with Dorian	90
Figure 4-3 Grid Grouper	91
Figure 6-1 CQL Schema.....	135
Figure 6-2 DCQL schema.....	150
Figure 6-3 Example DCQL Query.....	151
Figure 6-4 FederatedQueryProcessor Inheritance Model	154
Figure 6-5 FederatedQueryResultsClient Inheritance Model	156
Figure 6-6 Executing a Distributed Query.....	158
Figure 6-7 Executing a Non-Blocking Distributed Query	160
Figure 6-8 Destroying Query Results	161
Figure 6-9 Scheduling the Destruction of Query Results.....	162
Figure 6-10 Invoking the FederatedQueryEngine Locally	163
Figure 8-1 Overview of the architecture of the caGrid Workflow component	172
Figure 8-2 Workflow service state diagram	175

Tables

Table 1-1 Document Conventions	3
Table 9-1 Software prerequisites for GME	187

About This Guide

This section introduces you to the *caGrid 1.2 Programmer's Guide*. Topics in this section include

- *Purpose* on this page
- *Release Schedule* on this page
- *Audience* on this page
- *Topics Covered* on this page
- *Document Text Conventions* on this page
- *Credits and Resources* on page 3

Purpose

The cancer Biomedical Informatics Grid, or caBIG™, is a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open environment with common standards and shared tools. The current grid architecture of caBIG™ is dubbed caGrid. The software embodiment and corresponding documentation of this architecture constitute the caGrid release.

This guide describes the caGrid Application Programming Interfaces (APIs).

Release Schedule

This guide has been updated for the caGrid 1.2 release. It may be updated between releases if errors or omissions are found. The current document refers to the 1.2 version of caGrid, released in March 2008 by caBIG.

Audience

The primary audience of this guide is the caGrid service developer, client application developer, and service administrator. For additional information about installing and using caGrid, see the [caGrid 1.2 Programmer's Guide](#).

This guide assumes that you are familiar with the Java programming language and/or other programming languages, database concepts, and the Internet. If you intend to use caGrid resources in software applications, it assumes that you have experience with building and using complex data systems.

Topics Covered

This brief overview explains what you will find in each chapter and appendix of this guide.

- *About This Guide*, this chapter, provides an overview of the guide.
- *Chapter 1, Overview of caGrid*, on page 5 describes the three primary caGrid roles for which this guide is written.
- *Chapter 2, caGrid Release Structure*, on page 15 provides an overview and examples using the Introduce toolkit for service development.
- *Chapter 3, caGrid Metadata Infrastructure* on page 41 describes how to create caGrid data services.
- *Chapter 4, caGrid Security* on page 87 introduces the client applications for caGrid services.
- *Chapter 5, caGrid Service Developers* on page 121 describes the caGrid security infrastructure, which provides services and tools to administer and enforce security policy.
- *Chapter 6, caGrid Data Services* on page 133 describes the caGrid implementation of a workflow, which provides a grid service for submitting and running workflows that are composed of other grid services.
- *Chapter 7, WS-Enumeration*, on page 164 describes the client-side APIs for enumerations.
- *Chapter 8, Workflow Management Service* on page 171 describes the architecture and APIs for interacting with caGrid workflow.
- *Chapter 9, caGrid Global Model Exchange*, on page 187 describes the caGrid Global Model Exchange (GME).
- *Appendix A, References* on page 193 provides references relevant to caGrid.
- *Appendix B, Glossary* on page 199 defines acronyms, objects, tools and other terms related to caGrid.

Document Text Conventions

The following table shows how text conventions are represented in this guide. The various typefaces differentiate between regular text and menu commands, keyboard keys, and text that you type.

Convention	Description	Example
Bold	Highlights names of option buttons, check boxes, drop-down menus, menu commands, command buttons, or icons.	Click Search .
<u>URL</u>	Indicates a Web address.	http://domain.com
text in SMALL CAPS	Indicates a keyboard shortcut.	Press ENTER.

Convention	Description	Example
text in SMALL CAPS + text in SMALL CAPS	Indicates keys that are pressed simultaneously.	Press SHIFT + CTRL.
<i>Italics</i>	Highlights references to other documents, sections, figures, and tables.	See <i>Figure 4.5</i> .
<i>Italic boldface monospace</i> type	Represents text that you type.	In the New Subset text box, enter P<small>roprietary</small> P<small>roteins</small> .
Note:	Highlights information of particular importance.	Note: This concept is used throughout this document.
{ }	Surrounds replaceable items.	Replace {last name, first name} with the Principal Investigator's name.

Table 1-1 Document Conventions

Credits and Resources

caGrid 1.2 Programmer's Guide Development and Management Teams		
Development	Support (Systems, QA, Documentation)	Management
Scott Oster (Lead Architect) ¹	Aynur Abdurazik ⁷	Avinash Shanbhag (Product Manager) ⁵
David Ervin ¹	Chet Bochan ⁹	John Eisenschmidt ¹⁰
Ian Foster ²	Gavin Brennan ⁹	Michael Keller ⁶
Shannon Hastings ¹	Carolyn Kelley Klinger ⁸	David Wu ⁶
Tahsin Kurc ¹	Wei Lu ⁹	Peter Yan ⁷
Manav Kher ³	Ye Wu ⁵	
Stephen Langella ¹		
Ravi Madduri ²		
Kunal Modi ⁴		
Joshua Phillips ³		
Joel Saltz ¹		

<i>caGrid 1.2 Programmer's Guide Development and Management Teams</i>		
<i>Development</i>	<i>Support (Systems, QA, Documentation)</i>	<i>Management</i>
^{1.} Ohio State University - Biomedical Informatics Department	^{2.} University of Chicago/Argonne National Laboratory	^{3.} SemanticBits, LLC.
^{4.} Ekagra Software Technologies, Ltd.	^{5.} NCI - Center for Biomedical Informatics and Information Technology (CBIIT)	^{6.} Booz Allen Hamilton
^{7.} Science Application International Corporation (SAIC)	^{8.} Lockheed Martin Management System Designers	^{9.} Terrapin Systems LLC (TerpSys)
^{10.} 5AM Solutions		

<i>Other Acknowledgements</i>
GeneConnect – Project - Washington University
GridIMAGE – Project - Ohio State University
caBIO – Project - National Cancer Institute Center for Bioinformatics (NCICB)
caArray – Project - National Cancer Institute Center for Bioinformatics (NCICB)
caTRIP – Project – Duke Comprehensive Cancer Center
GenePattern – Project – Broad Institute
geWorkbench – Columbia University
caBiocondutor – Project – Fred Hutchinson Cancer Research Center

<i>Contacts and Support</i>		
NCICB Application Support	http://ncicbsupport.nci.nih.gov/sw/ Telephone: 301-451-4384 Toll free: 888-478-4423	

<i>LISTSERV Facilities Pertinent to caGrid</i>		
<i>LISTSERV</i>	<i>URL</i>	<i>Name</i>
cagrid_users-l@list.nih.gov	https://list.nih.gov/archives/cagrid_users-l.html	caGrid Users Discussion Forum

Chapter 1 Overview of caGrid

This chapter provides an overview of the cancer Biomedical Informatics Grid, or caBIG™, a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools.

Topics in this chapter include:

- Introduction on this page
- Standards Compliant on page 5
- Model Driven on page 6
- Semantically Discoverable on page 7
- Secure and Manageable on page 9
- Revolutionary Development on page 12

Introduction

The cancer Biomedical Informatics Grid, or caBIG™, is a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools. The underlying service oriented infrastructure that supports caBIG™ is referred to as caGrid. Driven primarily by scientific use cases from the cancer research community caGrid provides the core enabling infrastructure necessary to compose the Grid of caBIG™. It provides the technology that enables collaborating institutions to share information and analytical resources efficiently and securely, and allows investigators to easily contribute to and leverage the resources of a national-scale, multi-institutional environment

The caGrid 0.5 "test bed" infrastructure was released in September 2005, which included the initial set software tools to effectively realize the goals of caBIG™. The grid technologies and methodologies adopted for caBIG™, and implemented in caGrid, provide a loosely coupled environment wherein local providers are given freedom of implementation choices and ultimate control over access and management, but harmonize on community accepted virtualizations of the data they use, and make them available using standardized service interfaces and communication mechanisms. While caGrid enables numerous complex usage scenarios, in its simplest base, its goals are to: enable universal mechanisms for providing interoperable programmatic access to data and analytics to caBIG™, create a self-describe infrastructure wherein the structure and semantics of data can be programmatically determined, and provide a powerful means by which resources available in caBIG™ can be programmatically discovered and leveraged. Additional information about the caGrid 0.5 effort, and a good overview of the motivation of the grid approach of caBIG™, can be found in the Bioinformatics Journal article (<http://bioinformatics.oxfordjournals.org/cgi/content/full/22/15/1910>).

Building on the foundation of caGrid 0.5, caGrid 1.0 was extensively enhanced based on the feedback & input from the early adopters of the caGrid 0.5 infrastructure and additional requirements from the various caBIG™ Domain Workspaces. The release of caGrid version 1.0 represented a major milestone in the caBIG™ program towards achieving the program goals. It provided the implementation of the required core

services, toolkits and wizards for the development and deployment of community provided services, APIs for building client applications, and some reference implementations of applications and services available in the production grid. The caGrid 1.1 and caGrid 1.2 releases represent minor, backwards compatible releases of caGrid, with a focus on increased usability, bug fixes, and various feature enhancements. A detailed listing of the changes from caGrid 1.0 can be found in the release notes and on the project website.

Standards Compliant

As a primary principle of caBIG™ is open standards, caGrid is built upon the relevant community-driven standards of the World Wide Web Consortium (W3C <http://www.w3.org/>) and OASIS (<http://www.oasis-open.org>). It is also informed by the efforts underway in the Open Grid Forum (OGF <http://ogf.org/>). The Open Grid Forum (OGF) is a community of users, developers, and vendors leading the global standardization effort for grid computing. The OGF community consists of thousands of individuals in industry and research, representing over 400 organizations in more than 50 countries. As such, while the caGrid infrastructure is built upon the 4.0 version of the Globus Toolkit (GT4 <http://globus.org/toolkit/>), it shares a Globus goal to be programming language and toolkit agnostic by leveraging existing standards. Specifically, caGrid services are standard WSRF v1.2 services and can be accessed by any specification-compliant client.

caGrid 1.0 also represented an increased involvement in relevant working groups, standards bodies, and organizations involved in the standardization and adoption of grid technologies. The caGrid team consists of several members involved in both the development of the Globus toolkit, and authors on some of the relevant specifications. Furthermore, some of the components developed by caGrid have been published in peer-reviewed articles, have been vetted by the grid community in several invited talks, and are undergoing an incubation process to become part of the Globus toolkit itself.

Model Driven

Extending beyond the basic grid infrastructure, caBIG™ specializes these technologies to better support the needs of the cancer research community. A primary distinction between basic grid infrastructure and the requirements identified in caBIG and implemented in caGrid is the attention given to data modeling and semantics. caBIG™ adopts a model-driven architecture best practice and requires that all data types used on the grid are formally described, curated, and semantically harmonized. These efforts result in the identification of common data elements, controlled vocabularies, and object-based abstractions for all cancer research domains. caGrid leverages existing NCI data modeling infrastructure to manage, curate, and employ these data models. Data types are defined in caCORE UML and converted into ISO/IEC 11179 Administered Components, which are in turn registered in the Cancer Data Standards Repository (caDSR). The definitions draw from vocabulary registered in the Enterprise Vocabulary Services (EVS), and their relationships are thus semantically described. caGrid 1.0 represented a significant improvement in its leveraging of these technologies and the corresponding information they make available. caGrid 1.0 added support for grid service access to both the EVS and caDSR, and its new service metadata standards include significant additions of information extracted from the caDSR and EVS.

In caGrid, both the client and service APIs are object oriented, and operate over well-defined and curated data types. Clients and services communicate through the grid using respectively grid clients and grid service infrastructure. The grid communication protocol is XML, and thus the client and service APIs must transform the transferred objects to and from XML. This XML serialization of caGrid objects is restricted in that each object that travels on the grid must do so as XML which adheres to an XML schema registered in the Global Model Exchange (GME). As the caDSR and EVS define the properties, relationships, and semantics of caBIG™ data types, the GME defines the syntax of their XML materialization. Furthermore, caGrid services are defined by the Web Service Description Language (WSDL). The WSDL describes the various operations the service provides to the grid. The inputs and outputs of the operations, among other things, in WSDL are defined by XML schemas. As caBIG™ requires that the inputs and outputs of service operations use only registered objects, these input and output data types are defined by the XSDs which are registered in GME. In this way, the XSDs are used both to describe the contract of the service and to validate the XML serialization of the objects which it uses. Figure 1-1 details the various services and artifacts related to the description of and process for the transfer of data objects between client and service.

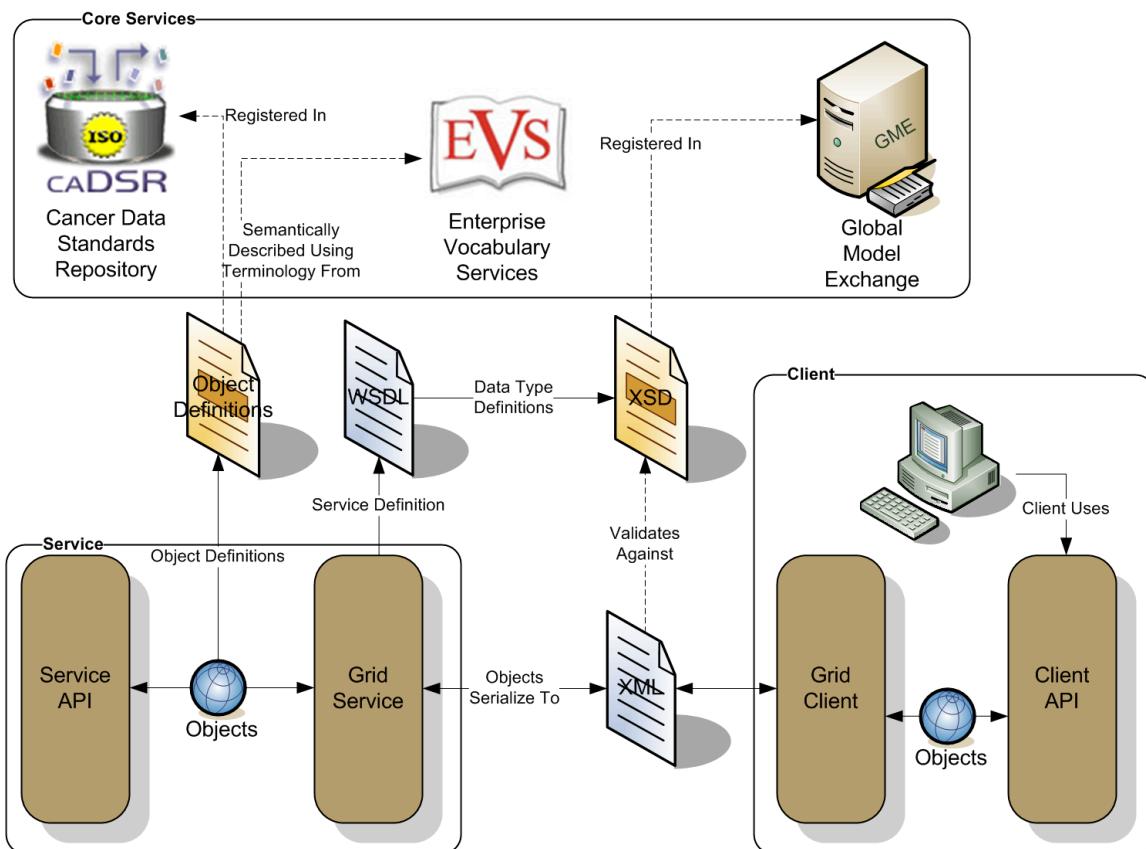


Figure 1-1 caGrid Data Description Infrastructure

Semantically Discoverable

As caBIG™ aims to connect data and tools from 50+ disparate cancer centers and many other institutions, a critical requirement of its infrastructure is that it supports the ability of

researchers to discover these resources. caGrid enables this ability by taking advantage of the rich structural and semantic descriptions of data models and services that are available. Each service is required to describe itself using caGrid standard service metadata. When a grid service is connected to the caBIG™ grid, it registers its availability and service metadata with a central indexing registry service (Index Service). This service can be thought of as the “yellow pages” and “white pages” of caBIG™. A researcher can then discover services of interest by looking them up in this registry. caGrid provides a series of high-level APIs and user applications for performing this lookup which greatly facilitate the discovery process.

As the Index Service contains the service metadata of all the currently advertised and available services in caBIG™, the expressivity of service discovery scenarios is limited only by the expressivity of the service metadata. For this reason, caGrid provides standards for service metadata to which all services must adhere. At the base is the common Service Metadata standard that every service in caBIG™ is required to provide. This metadata contains information about the service-providing cancer center, such as the point of contact and the institution’s name. Data Services, as a standardized type of caGrid services, also provide an additional Domain Model metadata standard. Both of these standards leverage the data models registered in caDSR and link them to the underlying semantic concepts registered in EVS. The Data Service Metadata details the domain model from which the Objects being exposed by the service are drawn. Additionally, the definitions of the Objects themselves are described in terms of their underlying concepts, attributes, attribute value domains, and associations to other Objects being exposed. Similarly, the common Service Metadata details the Objects, used as input and output of the services operations, using the same format as the Data Service metadata. In addition to detailing the Objects definitions, the Service Metadata defines and describes the operations or methods the service provides, and allows semantic concepts to be applied to them. In this way, all services fully define the domain objects they expose by referencing the data model registered in caDSR, and identify their underlying semantic concepts by referencing the information in EVS. The caGrid metadata infrastructure and supporting APIs and toolkits are defined with extensibility in mind, encouraging the development of additional domain or application specific extensions to the advertisement and discovery process.

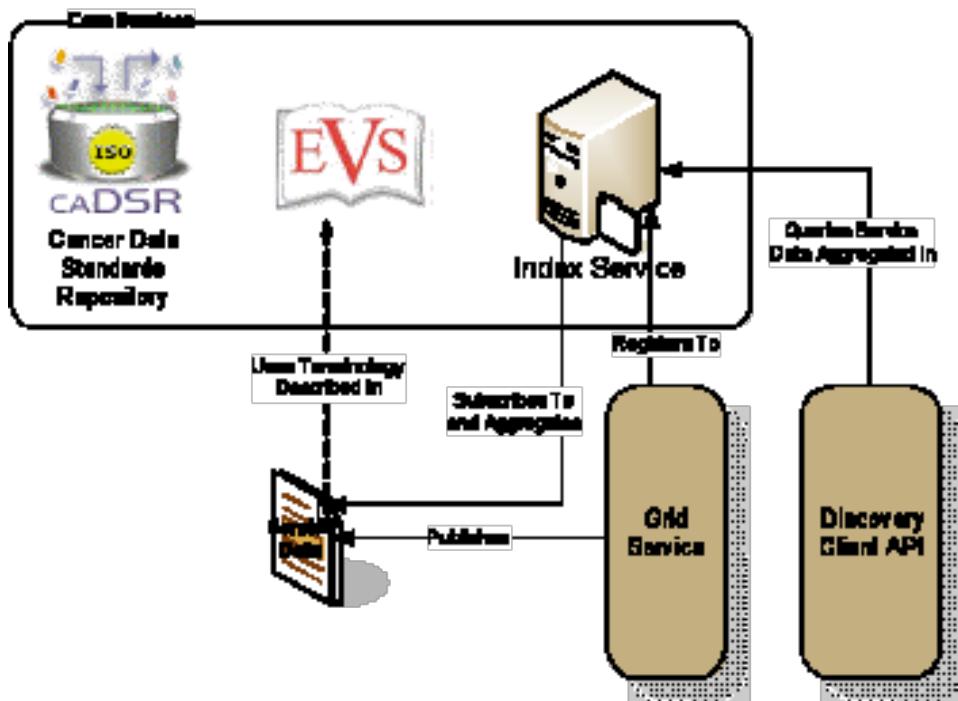


Figure 1-2 caGrid Discovery Overview

As shown in Figure 1-2, the caGrid discovery API and tools allow researchers to query the Index Service for services satisfying a query over the service metadata. That is, researchers can lookup services in the registry using any of the information used to describe the services. For instance, all services from a given cancer center can be located, data services exposing a certain domain model or objects based on a given semantic concept can be discovered, as can analytical services that provide operations that take a given concept as input.

Secure and Manageable

Security is an especially important component of caBIG™ both for protecting intellectual property and ensuring protection and privacy of patient related and sensitive information. caGrid 1.0 provided a complete overhaul of federated security infrastructure to satisfy caBIG™ security needs, incorporating many of the recommendations made in the caBIG™ Security White Paper, culminating in the creation of the Grid Authentication and Authorization with Reliably Distributed Services (GAARDs) infrastructure. GAARDs provides services and tools for the administration and enforcement of security policy in an enterprise Grid. caGrid 1.1 represented a major thrust to deploy GAARDs to the cancer research community, in that its release is timed and informed by the first set of policies and procedures created by the caBIG™ Security Working Group. The Security Working Group is a collaborative effort of the caBIG™ Architecture and Data Sharing and Intellectual Capital (DSIC) Workspaces that is intended to create and implement security policies to enable data sharing across the caBIG Federation. The initial policies in place for caGrid 1.1 formalized the envisioned Levels of Assurance for credentials in the grid, and detail the policies and practices of a credential provider adhering to the initial Level of Assurance (LOA1) which will govern the baseline credentials all caBIG™ participants may use. caGrid 1.2 adds support for credential delegation with the

Credential Delegation Service (CDS), and provides a solution for integrating grid security with web applications, in the form of the Web Single Sign On (WebSSO) project.

GAARDS was developed on top of the Globus Toolkit and extends the Grid Security Infrastructure (GSI) to provide enterprise services and administrative tools for:

- Grid user management
- Identity federation
- Trust management
- Group/VO management
- Access control policy management and enforcement
- Integration between existing security domains and the grid security domain

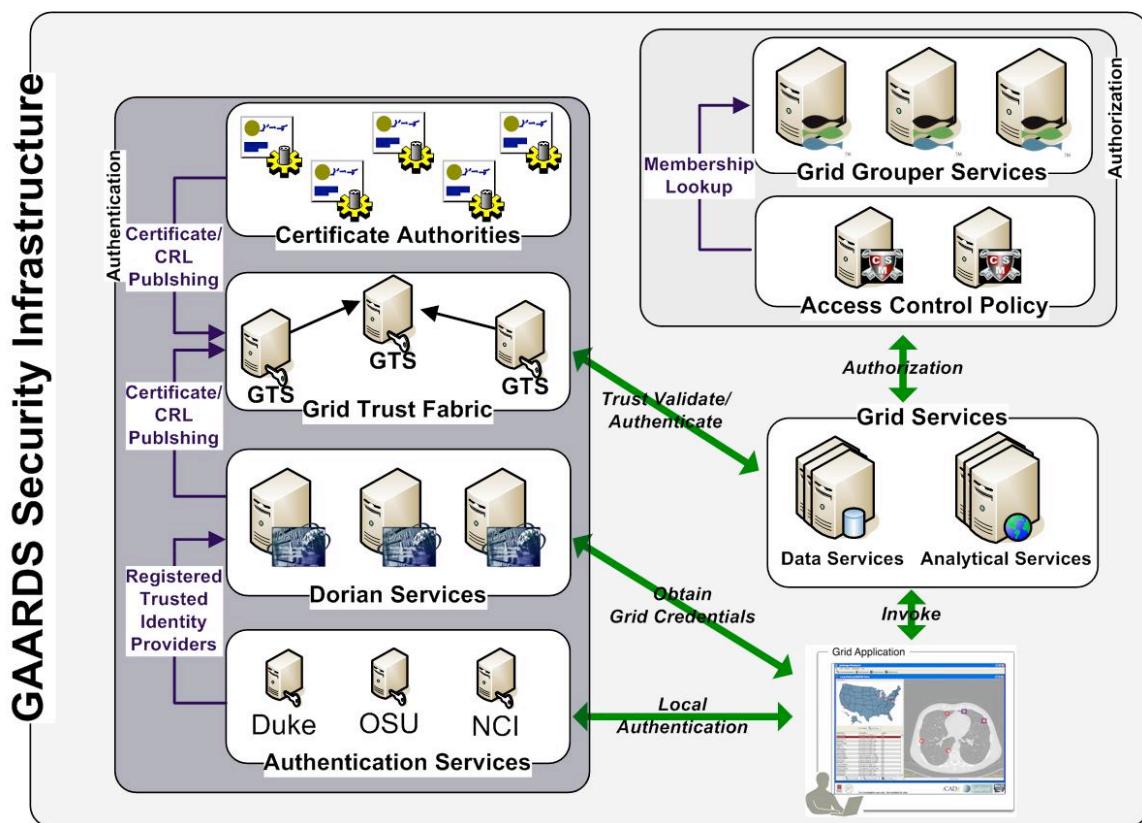


Figure 1-3 GAARDS Security Infrastructure

Figure 1-3 illustrates the GAARDS security infrastructure, in order for users/applications to communicate with secure services, they need grid credentials. Obtaining grid credentials requires having a Grid User Account. Dorian provides two methods for registering for a grid user account: 1) registering directly with Dorian 2) having an existing user account in another trusted security domain. In order to use an existing user account to obtain grid credentials, the existing credential provider must be registered in Dorian as a Trusted Identity Provider. It is anticipated that the majority of grid user accounts will be provisioned based on existing accounts. The advantages to this approach are: 1) users can use their existing credentials to access the grid 2) administrators only need to manage a single account for a given user. To obtain grid

credentials, Dorian requires proof (a digitally signed SAML assertion) that proves that the user locally authenticated. The GAARDS Authentication Service provides a framework for issuing SAML assertions for existing credential providers such that they may be used to obtain grid credentials from Dorian. The Authentication Service also provides a uniform authentication interface in which applications can be built. The process for obtaining grid credentials begins when the user/application authenticates with their local credential provider via the Authentication Service and obtains a SAML assertion as proof they authenticated. They then use the SAML assertion provided by the Authentication Service to obtain grid credentials from Dorian. Assuming the local credential provider is registered with Dorian as a trusted identity provider and that the user's account is in good standing, Dorian will issue grid credentials to the user. It should be noted that the use of the Authentication Service is not required; an alternative mechanism for obtaining the SAML assertion required by Dorian can be used. If a user is registered directly with Dorian and not through an existing credential provider, they may contact Dorian directly for obtaining grid credentials.

Once a user has obtained grid credentials from Dorian they may invoke secure services. Upon receiving grid credentials from a user, a secure service authenticates the user to ensure that the user has presented valid grid credentials. Part of the grid authentication process is verifying that grid credentials presented were issued by a trusted grid credential provider (i.e Dorian, other certificate authorities). The Grid Trust Service (GTS) maintains a federated trust fabric of all the trusted digital signers in the grid. Credential providers such as Dorian and grid certificate authorities are registered as trusted digital signers and regularly publish new information to the GTS. Grid services authenticate grid credentials against the trusted digital signers in a GTS.

Once the user has been authenticated, a secure grid service next determines if a user is authorized to perform what they requested. Grid services have many different options available to them for performing authorization. It is important to note that all authorizing decisions are made by the local provider, but GAARDS provides some services and tools which facilitate some common authorization mechanisms. The GAARDS infrastructure provides two approaches which can each be used independently or can be used together. It is important to note any other authorization approach can be used in conjunction with the GAARDS authentication/trust infrastructure. The Grid Grouper service provides a group-based authorization solution for the Grid, wherein grid services and applications enforce authorization policy based on membership to groups defined and managed at the grid level. Grid services can use Grid Grouper directly to enforce their internal access control policies. Assuming the authorization policy is based on membership to groups provisioned by Grid Grouper; services can determine whether a caller is authorized by simply asking grid grouper whether the caller is in a given group. The caCORE Common Security Module (CSM), an existing component many providers are already using, is a more centralized approach to authorization. CSM is a tool for managing and enforcing access control policy centrally. CSM supports access control policies which can be based on membership to groups in Grid Grouper. Grid services that use CSM for authorization simply ask CSM with a user can perform a given action. Based on the access control policy maintained in CSM, CSM decides whether or not a user is authorized. The grid services defer the authorization to CSM. CSM enforces its group based access control policy by asking Grid Grouper whether the caller is a member of the groups specified in the policy, and enforces any other local data access policies defined in CSM.

In order to support more complex use cases such as federated query, distributed workflow execution, and integration with existing web applications, a mechanism by which a user or service can allow another user or service to perform work on their behalf is needed. That is, as the authorization process is anchored from the authentication process (using grid credentials), there is a need to allow external users or services to assume ones identity for the purposes of authorizing with remote applications. For example, if a workflow is to be executed which queries secure data services, the service or tool executing the workflow needs to be able to execute those queries using the identity of the user requesting the workflow. It is for these purposes that the Credential Delegation Service was developed, which allows one (the delegator) to express a delegation policy, entitling a prescribed collection of other grid entities (the delegates) to assume the delegators identity for a limited lifetime. This capability is also leveraged by the Web Single Sign On (WebSSO) project, which provides a means by web users can navigate between participating applications without being challenged to provide login credentials. The framework provides the ability to leverage Dorian for login purposes, and can seamlessly make use of the CDS to provide the participating web application the ability invoke secure services with the grid identity of the currently logged in web user.

Revolutionary Development

caGrid 1.0 represented a complete rewrite of caGrid to better support the requirements and current standards. Building on lessons learned from caGrid 0.5 and feedback from the community, it provided a large number of additional features, services, and vast improvements in caGrid technologies beyond what is described above. One such example was the development of a unified grid service authoring toolkit, dubbed Introduce. Introduce is an extensible framework and graphic workbench which provides an environment for the development and deployment of caBIG™ compatible grid enabled data and analytical services. The Introduce toolkit reduces the service developer's responsibilities, by abstracting away the need to manage the low level details of the WSRF specification and integration with the Globus Toolkit, allowing them to focus on implementing their business logic. Developers with existing caBIG™ Silver compatible services need only follow simple a wizard-like process for creating the "adapter" between the grid and their existing system. At the same time, extremely complex and powerful new services can be created. All caGrid developed core services were implemented with the Introduce toolkit. caGrid 1.1 added the ability to migrate caGrid 1.0 Introduce services to caGrid 1.1 services, and provides the migration framework to handle all such future migrations. This capability was maintained for caGrid 1.2 and services created with either 1.0 or 1.1 can be upgraded to 1.2.

Another significant feature added with caGrid 1.0 is the service support for orchestration of grid services using the industry standard Business Process Execution Language (BPEL). caGrid provides a workflow management service, enabling the execution and monitoring of BPEL-defined workflows in a secure grid environment. It is expected this work will provide the groundwork for a large number of powerful applications, enabling the harnessing of data and analytics made available as grid services. Another such higher-level support service made available in caGrid 1.0, is the federated query infrastructure. The caGrid Federated Query Infrastructure provides a mechanism to perform basic distributed aggregations and joins of queries over multiple data services. Working in collaboration with the Cancer Translational Research Informatics Platform (caTRIP) project, a caBIG™ funded project, an extension to the standard Data Service

query language was developed to describe distributed query scenarios, as well as various enhancements to the Data Service query language itself. The Federated Query Infrastructure contains three main client-facing components: an API implementing the business logic of federated query support, a grid service providing remote access to that engine, and a grid service for managing status and results for queries that were invoked asynchronously using the query service.

Numerous improvements to the handling of large data sets and distributed information processing were also made. Support for the implantation of the WS-Enumeration (<http://www.w3.org/Submission/WS-Enumeration/>) standard has been implemented and added to the Globus Toolkit. This standard and its corresponding implementation provide the capability for a grid client to enumerate over results provided by a grid service (much like a grid-enabled *cursor*). This provides the framework necessary for clients to access large results from a service. This support has been integrated into the caGrid Data Service tooling providing a mechanism for iterating query results.

Additionally, an effort to standardize a “bulk data transport” interface for large data was started in caGrid 1.0, and improved in caGrid 1.1, which is intended to provide uniform mechanism by which clients may access data sets from arbitrary services. This initial work currently supports access via WS-Enumeration, WS-Transfer, and GridFTP. Additional enhancements and tooling are expected in a future release of caGrid, based on feedback from the user community. With caGrid 1.2, a lighter weight mechanism, called caGrid Transfer, was provided to enable http-based (both HTTP and GSI-enabled HTTPS) download and upload capabilities to services.

Lastly, caGrid 1.0 represented a significant improvement in quality of caGrid, as a considerable effort was placed on the development of unit, system, and integration testing. Several hundred unit tests are executed every time the caGrid code base is changed, and a variety of builds and tests are run each night. This effort was continued throughout the development and release of caGrid 1.1 and caGrid 1.2, and several hundred additional tests have been added. Interested users may view results of these tests on a centralized dashboard (<http://quality.cagrid.org/caGrid-1.0/Dashboard/>), execute these test frameworks locally, or leverage the testing framework during the development of their own services.

Chapter 2 caGrid Release Structure

This chapter describes the basic layout of the caGrid release.

Topics in this chapter include:

- Overview on this page
- caGrid Projects on this page

Overview

caGrid is released as a source release, as well as with an automated installer. In order to use any feature of caGrid or to develop applications with it, you must build caGrid. The installer will do this automatically for you, but if you are using the source release, you must build it yourself. You can find detailed instructions on building caGrid on the caGrid wiki: <http://www.cagrid.org/wiki/CaGrid:How-To:Build>. Generally you just need to type the following command from the caGrid directory:

```
ant all
```

The caGrid release is oriented around a number of individual projects and the build process manages inter-project dependencies. Each project provides a specific set of functionality, and is self-contained once caGrid is built. That is, once caGrid is built, each of the projects can be used independently. For example, if you are only interested in Introduce, you can safely copy around caGrid/projects/introduce as a standalone copy of Introduce. The same is true for core services; once caGrid is built, all their dependencies are copied into the lib and ext/lib directories of the service.

caGrid Projects

The following is a list of the projects provided in the caGrid release and a brief description of each.

- advertisement
 - **Description:** This project contains the APIs needed for programmatically registering a service with the Index Service.
 - **Type:** Service APIs
 - **Usage:** These APIs are automatically integrated into Introduce-created services, and so are generally not needed by caGrid users, unless they are developing their own services without using Introduce.
- authentication-service
 - **Description:** This project contains the Authentication Service which provides a standardized service interface for exposing Dorian Identity Providers on the grid.
 - **Type:** Service
 - **Usage:** Anyone wishing to integrate an institutional user management system as a Dorian Identity Provider, can implement this service. Similarly, applications which provide log-in capabilities should use the client APIs provided by this service.

- authz
 - **Description:** This project contains the various authorization components caGrid provides for applying local authorization policy to services.
 - **Type:** Service APIs
 - **Usage:** Introduce provides integration with these components, but they can also be used standalone by service developers.
- bulkDataTransfer
 - **Description:** This project contains the Bulk Data Transfer service and Introduce Extension, which is intended as a common interface for returning large data to a client from a service.
 - **Type:** Introduce Extension
 - **Usage:** Introduce provides integration with this component for service developers, but clients can use its client APIs for interacting with services that support this interface.
- cabigextensions
 - **Description:** This project contains a variety of Introduce extensions which customize Introduce for caBIG needs. These extensions include the caGrid metadata, ws-Enumeration, metadata viewers, and XMI based XSD creation extensions.
 - **Type:** Introduce Extension
 - **Usage:** Introduce provides integration with these components and they generally are not needed by caGrid users.
- cadsr
 - **Description:** This project contains the caDSR Grid Service, which provides access to registered models and the capability to generate service metadata.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the caDSR Grid service, as well as the code and configuration necessary to deploy a local instance.
- cds
 - **Description:** This project contains the Credential Delegation Service (CDS), which enables users/services (delegator) to delegate their Grid credentials to other users/services (delegatee) such that the delegatee(s) may act on the delegator's behalf.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the CDS Grid service, as well as the code and configuration necessary to deploy a local instance.
- core
 - **Description:** This project contains a common set of utilities and APIs used by many caGrid projects.

- **Type:** APIs
- **Usage:** Most caGrid users will find the utilities provided by the project to be of general use in developing applications and services with caGrid. Some examples in this document use some of these APIs, but a complete reference can be found in the javadoc.
- cql
 - **Description:** This project contains the schemas and beans for the caGrid standard data service query languages.
 - **Type:** APIs
 - **Usage:** Those wishing to programmatically work with CQL can use the libraries provided by this project.
- data
 - **Description:** This project contains the core data service infrastructure, tooling and utilities, as well as the definition and APIs for CQL.
 - **Type:** APIs/Service APIs
 - **Usage:** Most of the service code provided by this project is already added to data services which are created with Introduce, but the utilities it provides are of general use for custom data service implementations. Those developing applications which make use of data services or CQL will want to use the client APIs and utilities this project provides.
- dataExtensions
 - **Description:** This project contains the Introduce extensions which add support for the data service infrastructure to Introduce.
 - **Type:** Introduce Extension
 - **Usage:** Introduce provides integration with these components and they generally are not needed by caGrid users. However, those developing custom data service extensions may find this a useful starting point.
- discovery
 - **Description:** This project contains the APIs and code examples for discovering services from the Index Service. Examples are provided elsewhere in this document.
 - **Type:** APIs
 - **Usage:** Those developing applications which make use of discovery may use the client APIs provided by this project.
- dorian
 - **Description:** This project contains the Dorian Service which provides a user and credential management service. Dorian is described elsewhere in this document.
 - **Type:** Service
 - **Usage:** The client APIs and utilities provided by this project can be used to provide authentication capabilities to applications. It also provides an

administrative GUI, but its functionality can also be used through the general security GUI. It also contains the code and configuration necessary to deploy a local instance of Dorian.

- evs
 - **Description:** This project contains the EVS Grid Service, which is detailed later in this document.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the EVS Grid service, as well as the code and configuration necessary to deploy a local instance.
- fqp
 - **Description:** This project contains the Federated Query Processor (FQP) Grid Service and engine, which are detailed later in this document.
 - **Type:** Service / APIs
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the FQP Grid service, the code and configuration necessary to deploy a local instance, and the APIs for interacting with the FQP query engine locally in an application.
- gaards-ui
 - **Description:** This project contains the configuration files and libraries necessary for the unified security GUI.
 - **Type:** APIs/Tools
 - **Usage:** This project can be used to launch the unified security GUI (in addition to the main caGrid build file), and provides the graphical interfaces to GAARDS.
- gme
 - **Description:** This project contains the Global Model Exchange (GME) Grid Service, which is detailed later in this document.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the GME Grid service, as well as the code and configuration necessary to deploy a local instance.

- grape
 - **Description:** This project contains the Grid Application Environment (GrApE), which is a framework for developing component-oriented applications.
 - **Type:** APIs/Tools
 - **Usage:** Various caGrid GUI tools (such as the security administrative portal and workflow GUIs) leverage these APIs as a framework for developing graphical applications. APIs and sample configurations can be found for those wishing to build on this framework for “thick client” applications.
- graph
 - **Description:** This project contains an API for graphically rendering UML models.
 - **Type:** APIs/Tools
 - **Usage:** Those wishing to render UML models in Swing can leverage this library.
- gridca
 - **Description:** This project contains various APIs and command line tools for working with Certificates.
 - **Type:** APIs/Tools
 - **Usage:** Most caGrid users will not need to work with these APIs or command line tools unless they are developing low level security components. Most of the caGrid security components use these APIs internally.
- gridftpauthz
 - **Description:** This project contains the code necessary to hook in caGrid authorization to GridFTP..
 - **Type:** Service APIs
 - **Usage:** Service developers wishing to make use of GridFTP and provide an integrated security solution can make use of these APIs to secure the GridFTP server with an authorization callout which the service can control. A GridGrouper and Database implementation are provided.
- gridgrouper
 - **Description:** This project contains the GridGrouper Grid Service, which is detailed later in this document.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the GridGrouper Grid service, as well as the code and configuration necessary to deploy a local instance. It also provides an administrative GUI, but its functionality can also be used through the general security GUI.

- gts
 - **Description:** This project contains the Grid Trust Service (GTS) Grid Service, which provides the capability to manage the trust fabric of the grid.
 - **Type:** Service
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the GTS Grid service, as well as the code and configuration necessary to deploy a local instance. It also provides an administrative GUI, but its functionality can also be used through the general security GUI.
- index
 - **Description:** This project contains the code necessary to deploy the Globus Index Service.
 - **Type:** Service
 - **Usage:** Those wishing to deploy a local instance of the Index Service may use this project.
- installer
 - **Description:** This project contains the code used to create the caGrid installer.
 - **Type:** APIs/Tools
 - **Usage:** caGrid users generally do not need to use this project, though it could be used to create a specialized installer.
- introduce
 - **Description:** This project contains the Introduce graphical service development toolkit.
 - **Type:** APIs/Tools
 - **Usage:** Service developers can launch Introduce from this project (or using the ant target in the main caGrid build file). Those wishing to develop extensions to Introduce can also make use of its APIs and configuration examples.
- metadata
 - **Description:** This project contains the schemas and beans for the caGrid standard metadata formats.
 - **Type:** APIs
 - **Usage:** Those wishing to programmatically work with caGrid metadata can use the libraries provided by this project.
- metadatautils
 - **Description:** This project contains utilities for working with caGrid metadata and retrieving it from services.
 - **Type:** APIs

- **Usage:** Those wishing to programmatically work with caGrid metadata and access it from services can use the libraries provided by this project.
- opensaml
 - **Description:** This project contains a repackaged version of opensaml (to avoid conflicts with the version used in Globus).
 - **Type:** APIs
 - **Usage:** This is used internally by Dorian and the Authentication Service, and is not generally usable by caGrid users.
- sdkQuery
 - **Description:** This project contains the caCORE SDK version 3.1 implementation of a CQL Query processor.
 - **Type:** Service APIs
 - **Usage:** These APIs are automatically integrated into Introduce-created, SDK-backed, data services, and so are generally not needed by caGrid users, unless they wish to create a modified version of the query processor.
- sdkQuery32
 - **Description:** This project contains the caCORE SDK version 3.2.x implementation of a CQL Query processor.
 - **Type:** Service APIs
 - **Usage:** These APIs are automatically integrated into Introduce-created, SDK-backed, data services, and so are generally not needed by caGrid users, unless they wish to create a modified version of the query processor.
- sdkQuery4
 - **Description:** This project contains the caCORE SDK version 4.0 implementation of a CQL Query processor.
 - **Type:** Service APIs
 - **Usage:** These APIs are automatically integrated into Introduce-created, SDK-backed, data services, and so are generally not needed by caGrid users, unless they wish to create a modified version of the query processor.
- service-security-provider
 - **Description:** This project contains the client service implementation of the operation used to configure operation security settings based on caGrid standard security metadata.
 - **Type:** Service APIs
 - **Usage:** Introduce-generated clients already make use of these libraries, so they are not explicitly needed by most service developers or clients. caGrid users developing low level generic service consuming applications may find use for them. Also, services wishing to override the default behavior of clients may also modify this code.
- service-tools

- **Description:** This project contains a variety of miscellaneous APIs for working with events, groups, and databases.
- **Type:** Service/APIs
- **Usage:** This project contains APIs which can be used in service implementations; it is leveraged by some of the security services.
- syncgts
 - **Description:** This project contains the SyncGTS Grid Service and client APIs, which provide the capability to manage a node's trust certificates by querying one or more GTS services.
 - **Type:** Service/APIs
 - **Usage:** This project contains the client APIs, code examples, and client tooling for managing a node's trusted certificates as well as the code and configuration necessary to deploy a local SyncGTS service.
- testing
 - **Description:** This project contains APIs and resources used for testing services.
 - **Type:** APIs
 - **Usage:** This project contains the APIs, and resources which may be of use to service developers wishing to create functional tests.
- tools
 - **Description:** This project, expected to contain additional utilities, currently contains a framework for validating the functionality of some core services.
 - **Type:** APIs/Tools
 - **Usage:** This project may be of use to extends for those wishing to validate the functionality of services they develop.
- transfer
 - **Description:** caGrid Transfer is an Introduce Extension and supporting transfer service which will enable services and clients to download and upload bulk data easily and efficiently to and from grid services and clients.
 - **Type:** Introduce Extension/APIs
 - **Usage:** Introduce provides integration with this component for service developers, but clients can use its client APIs for interacting with services that support this interface. It also contains the supporting web application which needs to be deployed to make use of it in services.
- websso
 - **Description:** This project provides a comprehensive, Single Sign On solution for web application using caGrid's GAARDS Framework.
 - **Type:** Web Application
 - **Usage:** This project contains a web application which is the server component of websso.

- websso-client
 - **Description:** This project provides a comprehensive, Single Sign On solution for web application using caGrid's GAARDS Framework.
 - **Type:** Web Application/APIs
 - **Usage:** This project contains APIs and servlet filters which represent the client component of websso.
- wizard
 - **Description:** This project contains a simple wizard framework for building swing wizard-style interfaces.
 - **Type:** APIs
 - **Usage:** This is used internally by the caGrid installer, but may be of use to others doing similar functions.
- workflow
 - **Description:** This project contains the Workflow Factory Grid Service, which provides the capability to execute BPEL-based workflows.
 - **Type:** Service/Tools
 - **Usage:** This project contains the client APIs, code examples, and client tooling for interacting with the workflow service, as well as the code and configuration necessary to deploy a local instance. It also provides a GUI for executing and monitoring workflows.
- wsEnum
 - **Description:** This project contains the schemas, APIs, and utilities for implementing and working with the WS-Enumeration standard.
 - **Type:** APIs
 - **Usage:** Clients wishing to use generic WS-Enumeration clients may use the client libraries from this project.

Multiple Grid Support

Since caGrid 1.1, the release provides the ability to be easily reconfigured to access multiple different grid environments. That is, when clients and applications access "the grid" they are configured with the appropriate settings (e.g. service addresses, security settings, etc) for the appropriate grid. The most common use for this capability is to easily target a development or training grid when learning to use caGrid or while developing applications and services, and then retargeting the official production grid for production development or deployment.

Process

Once caGrid is built, it is already configured to target a default grid. If you are using a development checkout of caGrid, this will be the development grid. If you are using a release, this will be the production grid. You can find details about which grid you are currently targeting at anytime by typing the following command from the caGrid directory:

```
ant configureHelp
```

Running this command should output something similar to the following.

```
configureHelp:  
[echo] ======  
[echo] |      CONFIGURED TO USE GRID: nci_prod-1.2 AT: ${target.grid.configure.time}  
[echo] |  
[echo] |      THE VALID TARGET GRIDS ARE:  
[echo] |      - nci_dev-1.1  
[echo] |      - nci_dev-1.2  
[echo] |      - nci_prod-1.1  
[echo] |      - nci_prod-1.2  
[echo] |      - nci_qa-1.1  
[echo] |      - nci_qa-1.2  
[echo] |      - nci_stage-1.1  
[echo] |      - nci_stage-1.2  
[echo] |      - osu_dev-1.1  
[echo] |      - osu_dev-1.2  
[echo] |      - training-1.1  
[echo] |      - training-1.2  
[echo] |-----  
[echo] |      NOTE: To use a different target grid, set property 'target.grid'  
[echo] |      to the grid's name, and run the 'configure' target.  
[echo] |  
[echo] |      For example, type:  
[echo] |          ant -Dtarget.grid=nci_prod-1.2 configure  
[echo] ======  
  
BUILD SUCCESSFUL  
Total time: 2 seconds
```

The first line shows the current target grid, and when caGrid was configured to use it. Next the available target grids are listed. Finally, brief instructions are given for how to change do a different grid.

To use the training grid, the following command should be run from the caGrid directory:

```
ant -Dtarget.grid=training-1.2 configure
```

Note: In the caGrid 1.1 release, you would use: `ant -Dtarget.grid=training configure`

Running this command will:

- run through a process of reconfiguring all of the projects which have some dependency on the grid
- remove any locally stored preferences which are grid specific (such as used by Introduce and the security UIs)
- reinstall all caGrid Introduce extensions into Introduce
- synchronize your local machine with the appropriate trust fabric
- display the configuration information which was just applied (as is shown above from the `configureHelp` target)

Note: If the command is not successful, make sure you are using a `target.grid` as specified in the list of valid grids from the `configureHelp` target, and you have previously built caGrid (`ant all`). Also note that cleaning and rebuilding caGrid does not remove these settings; they must be reset with the `configure-clean` command.

To change back to the default grid at any time, you may just type the following command from the caGrid directory:

```
ant configure-clean configure
```

Issues

If you are building caGrid 1.2, you need to make sure to use a 1.2 grid. If you use a 1.1 grid, you may see an error like this:

```
[ivy:resolve] :: problems summary ::
[ivy:resolve] ::::: WARNINGS
[ivy:resolve] :::::::::::::::::::::
[ivy:resolve] :::: UNRESOLVED DEPENDENCIES :::
[ivy:resolve] :::::::::::::::::::::
[ivy:resolve] :: caGrid#target_grid;training-1.1: configuration(s) not
found in caGrid#target_grid;training-1.1: cds-ui. It was required from
caGrid#gaards-ui;1.2-dev target_grid
[ivy:resolve] :::::::::::::::::::::
[ivy:resolve]
[ivy:resolve]
[ivy:resolve] :: USE VERBOSE OR DEBUG MESSAGE LEVEL FOR MORE DETAILS
```

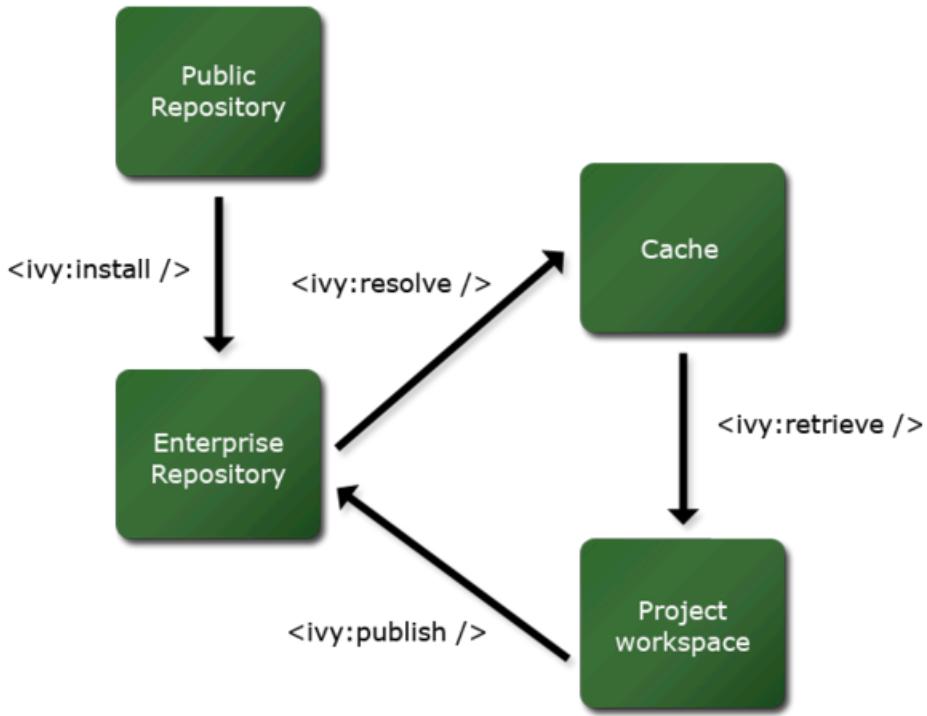
This is because caGrid 1.2 has new services (such as the CDS) that some components depend on, which did not exist in caGrid 1.1.

Depending on caGrid

caGrid is released as a source distribution, and as a graphical installer. Those leveraging caGrid are strongly encouraged to still use a caGrid release, however, when developing applications which use caGrid libraries, it can be difficult to know which caGrid libraries (or external libraries caGrid needs) you need for what you want to do with caGrid. As of caGrid 1.2, this information is formally managed and documented using the popular dependency management framework call Ivy. Ivy is a sub-project of Ant, and provides a number of custom Ant tasks focused solely on creating an enterprise dependency manager. caGrid transparently makes use of Ivy for its build process to manage its internal and external dependencies. Those not wishing to use Ivy are unaffected by this fact, but for those who do, it provides great advantages in determining exactly which libraries are needed for which tasks.

Ivy Introduction

The Ivy Documentation itself is quite good, and those new to Ivy are encouraged to review it. However, the main idea is that dependencies are described by each project in an Ivy file. An Ant-based build process (such as caGrid's process) is developed to use Ivy, and configures Ivy's settings, which control things like how Ivy should find dependencies. Then when a given project is built by the Ant build process, first its dependencies are resolved and retrieved from a repository into the project (such as into its lib directory). The project can then be built as it normally would, and optionally could publish itself to such a repository. A graphical representation of this process is shown below.



The use of Ivy does not necessarily need to affect your project's normal build process. In fact, only the main caGrid build process uses Ivy, and each caGrid project's build process is completely independent of Ivy.

Depending on caGrid using Ivy

There are several options for leveraging the power of Ivy when developing a project that depends on caGrid. This includes different ways of integrating Ivy directly into your project's build framework, just using Ivy to bootstrap your build by using it to initially copying what you need, or even just looking at the detailed reports Ivy generates and copying libraries "the old fashioned way." In the following sections, some of these options are discussed in the framework of a running example.

A simple example project appropriate for caGrid

For the purposes of demonstrating Ivy, a sample project which uses the Discovery APIs of caGrid, is explained below. The full source code for the example can be downloaded here or can be checked out from the *Demos* module of caGrid's CVS.

The sample project contains a single source file, contained in the `src` folder, and its Ant build process (defined in `build.xml`) expects to find the libraries it needs in the `lib` directory, and in the `lib` directory of the Globus installation (it expects the `GLOBUS_LOCATION` environment variable to be set). The build process compiles into the `build` directory, and contains an `an` target to run the main method of the source code (which makes various discovery queries). Thus far, this is a fairly typical project, and makes no reference to Ivy or a caGrid installation.

Here is the fairly simple build process for this project:

```

<project name="ivyexamples-discovery" default="run">

    <property name="lib.dir" value="lib" />
    <property name="build.dir" value="build" />
    <property name="src.dir" value="src" />

    <property environment="env" />
    <property name="globus.dir" value="${env.GLOBUS_LOCATION}" />
    <fail unless="env.GLOBUS_LOCATION" message="You must set the environment
variable GLOBUS_LOCATION to the directory in which you have Globus installed."
/>

    <path id="lib.path.id">
        <fileset dir="${lib.dir}">
            <include name="*.jar" />
        </fileset>
        <fileset dir="${globus.dir}/lib">
            <include name="*.jar" />
            <exclude name="caGrid-*.jar" />
        </fileset>
    </path>
    <path id="run.path.id">
        <path refid="lib.path.id" />
        <path location="${build.dir}" />
        <path location="${globus.dir}" />
    </path>

    <target name="run" depends="clean" description="--> compile and run the
project">
        <mkdir dir="${build.dir}" />
        <javac srcdir="${src.dir}" destdir="${build.dir}"
classpathref="lib.path.id" />
        <java classpathref="run.path.id"
classname="org.cagrid.ivyexamples.discovery.DiscoveryExamples" />
    </target>

    <target name="clean" description="--> cleans the project">
        <delete dir="${build.dir}" failonerror="false" />
    </target>
</project>
```

If we run this project as is, it will attempt to compile the project and generate a number of compiler errors. If we were to place all of the caGrid dependencies needed to compile this project's source code, it would run correctly. This brings us to the topic at hand; how to depend on caGrid.

Integrating Ivy into the project

As mentioned above, there are numerous ways Ivy can be used to solve our problem of depending on caGrid. If we actually want to use Ivy directly in our project, we need to describe to Ivy what our dependencies are. The approach taken in this example is to isolate the Ivy-related portions of the build (such as retrieving dependencies), from the "standard" build targets and build process (those defined above). In this way, we will actually create additional Ant build files which import our standard build, and add the Ivy support. We will use these additional targets when we need to leverage Ivy, but can use the standard build file for the normal compile and run process. The advantage to this approach is that we can actually just exclude the Ivy-related files if we were to create a distribution of our project which has all the jars it needs in its `lib` directory. However, if

we wanted to actually integrate Ivy into our build process, we could have just as easily done that as well.

The first step to integrating Ivy into our project is to actually define the project in terms of what it is, how you can depend on it, and what dependencies it has. This is done by creating an [Ivy file \(ivy.xml\)](#) for the project. A detailed explanation of these files is out of scope for this discussion, but suffice it to say the one used for this project is quite simple. It just indicates that version 1.2 of the *discovery* module of caGrid is needed. For more information about Ivy files, see the [Ivy documentation](#). Additionally, the caGrid release itself contains hundreds of Ivy files which can be used as examples. You can also look at these online in the [caGrid remote Ivy repository](#).

The Ivy file used for this project is shown below:

```
<ivy-module version="2.0">
  <info organisation="org.cagrid.ivyexamples" module="discovery"/>
  <dependencies>
    <dependency org="caGrid" name="discovery" rev="1.2"/>
  </dependencies>
</ivy-module>
```

Now that we have defined the project and its needs in a way Ivy understands, we need to tell Ivy what we want it to do for us; this is where we have choices. Ivy retrieves project dependencies from a repository. We have to tell Ivy what repository it should used to locate the caGrid dependencies our project needs. The sample project actually demonstrates two different ways to do this (explained in the next sections).

Using Ivy to extract libraries from a local caGrid installation

In the first example, we will configure Ivy to copy caGrid dependencies from a local build of the caGrid source distribution. We will refer to this installation's location with the environment variable `CAGRID_LOCATION`. As mentioned before, the approach we will be taking for integrating Ivy is to create another Ant build file, which extends our project's regular build process. In this second build file, the most important target we will create is the `ivy-resolve` target, which will instruction Ivy to copy the dependencies we need into our `lib` directory. We will also define the supporting targets of `ivy-clean` which will remove these copied dependencies, and `ivy-report` which will tell Ivy to create us a report of what it has done for us. We will create an `all` target, which simply calls all three of these targets, and make it the default for the project.

The build file is shown here:

```
<project name="ivyexamples-discovery-local-ivy" default="all"
xmlns:ivy="antlib:org.apache.ivy.ant">

  <import file="build.xml" />

  <property name="caGrid.dir" value="${env.CAGRID_LOCATION}" />
  <fail unless="env.CAGRID_LOCATION" message="You must set the environment
variable CAGRID_LOCATION to the directory in which you have caGrid installed
(the caGrid directory)." />

  <taskdef resource="org/apache/ivy/ant/antlib.xml"
uri="antlib:org.apache.ivy.ant">
    <classpath>
      <fileset dir="${lib.dir}" includes="ivy-*.jar" />
    </classpath>
  </taskdef>
```

```

<property name="repository.dir" location="${caGrid.dir}/repository" />
<property name="integration.repository.dir"
location="${caGrid.dir}/integration-repository" />
<ivy:settings id="ivy.instance"
file="${caGrid.dir}/antfiles/ivy/ivysettings.xml" override="true" />

<target name="ivy-resolve" description="--> retreive dependencies with ivy">
    <ivy:retrieve type="jar" />
</target>

<target name="ivy-report" depends="ivy-resolve" description="--> generates a
report of dependencies">
    <ivy:report todir="${build.dir}" />
</target>

<target name="ivy-clean" description="--> cleans all the libraries which
were retrieved for the project">
    <delete includeemptydirs="true" failonerror="false">
        <fileset dir="${lib.dir}">
            <include name="**/*" />
            <exclude name="ivy*.jar" />
        </fileset>
    </delete>
    <ivy:cleancache />
</target>

<target name="all" depends="ivy-clean,ivy-resolve,ivy-report"/>
</project>

```

Aside from the aforementioned Ant targets, there are some other aspects of the build file which need explanation. The first is that we define the new Ant tasks Ivy provides with an Ant `taskdef` and define a namespace (`ivy`) for these new tasks. This process is explained in the Ivy Ant documentation. Also of importance is how we tell Ivy about our repository. This is done by configuring Ivy with the `ivy:settings` task. The use of this task is defined in [the relevant section of the Ivy documentation, but of notice is that we are just pointing to the settings file which caGrid uses in it's build process. This settings file actually needs two variables to be set, which we can do with simple Ant properties. The first is the `repository.dir` property, which is where the external libraries caGrid needs are stored, and the second is the `integration.repository.dir` which is where caGrid projects publish themselves when built such that other caGrid projects can depend on them.

Once Ivy is configured, you'll note that the actual Ant targets are quite simple. To get the jars we need, we just need to call `ivy:retrieve`. It is worth noting that we are taking advantage of some of the default settings of Ivy, but these can always be overridden. For example, Ivy by default will copy jars into the `lib` directory, but that [can be changed](#).

To actually run this process and copy the dependencies we need into our `lib` directory, we just need to type:

```

ant -f local-ivy-build.xml
An example output of this can be see below:
[oster@gcity ~...ivyexamples/discovery]$ ant -f local-ivy-build.xml
Buildfile: local-ivy-build.xml
[ivy:settings] :: Ivy 2.0.0-beta2 - 20080225093827 :: 
http://ant.apache.org/ivy/ ::
[ivy:settings] :: loading settings :: file =
D:\projects\caBIG\ReleaseWork\caGrid-1.2\caGrid-1.2-
rc6\caGrid\antfiles\ivy\ivysettings.xml

```

```

ivy-clean:

ivy-resolve:
[ivy:retrieve] :: resolving dependencies :: 
org.cagrid.ivyexamples#discovery;working@gcity
[ivy:retrieve] confs: [default]
[ivy:retrieve] found caGrid#discovery;1.2 in caGrid-local-fs-fs
[ivy:retrieve] found caGrid#core;1.2 in caGrid-local-fs-fs
[ivy:retrieve] found castor#castor;0.9.9 in caGrid-fs-fs
[ivy:retrieve] found jdom#jdom;1.0 in caGrid-fs-fs
[ivy:retrieve] found caGrid#metadatautils;1.2 in caGrid-local-fs-fs
[ivy:retrieve] found caGrid#metadata;1.2 in caGrid-local-fs-fs
[ivy:retrieve] found jasciidammit#jasciidammit;1.1 in caGrid-fs-fs
[ivy:retrieve] found clarkware#jdepend;2.7 in caGrid-fs-fs
[ivy:retrieve] :: resolution report :: resolve 474ms :: artifacts dl 35ms
-----
|      |      modules      ||  artifacts  |
|  conf  |  number| search|dwnlded|evicted||  number|dwnlded|
-----
|  default  |  8  |  8  |  0  |  0  ||  17  |  0  |
-----
[ivy:retrieve] :: retrieving :: org.cagrid.ivyexamples#discovery
[ivy:retrieve] confs: [default]
[ivy:retrieve] 12 artifacts copied (2777kB/240ms)

ivy-report:
[ivy:report] Processing d:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\.cache\org.cagrid.ivyexamples-discovery-
default.xml to D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build\org.cagrid.ivyexamples-discovery-
default.html
[ivy:report] Processing d:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\.cache\org.cagrid.ivyexamples-discovery-
default.xml to D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build\org.cagrid.ivyexamples-discovery-
default.graphml

all:

BUILD SUCCESSFUL
Total time: 2 seconds

```

You'll note Ivy tells it copied 12 artifacts into our project, and we can view the report in our `build` directory.

You can now run the regular build by typing:

`ant`

You should see output similar to this:

```

Buildfile: build.xml

clean:
[delete] Deleting directory D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build

run:
[mkdir] Created dir: D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build
[javac] Compiling 1 source file to D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build
[java] =====

```

```

[java] Querying by All Services
[java] =====
[java]     0) Address: https://140.254.80.54:6443/wsrf/services/cagrid/Dorian
[java]     1) Address: https://cagrid-
cccfu.medctr.ad.wfubmc.edu:8443/wsrf/services/cagrid/Caaers
[java]     2) Address:
http://165.112.132.173:18080/wsrf/services/cagrid/CaArraySvc
[java]     3) Address:
http://165.112.132.241:18080/wsrf/services/cagrid/CaArraySvc
[java]     4) Address:
http://128.252.227.214:18080/wsrf/services/cagrid/CaArraySvc
[java]     5) Address:
https://192.168.200.198:8443/wsrf/services/cagrid/CoffeeMaker
[java]     6) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/EVSGridService
[java]     7) Address: https://10.144.36.26:8443/wsrf/services/cagrid/Dorian
[java]     8) Address:
http://140.254.80.50:50101/wsrf/services/cagrid/GridImageResultService
[java]     9) Address: https://cagrid-
dorian.nci.nih.gov:8443/wsrf/services/cagrid/Dorian
[java]    10) Address:
https://localhost:443/wsrf/services/cagrid/QueryService
[java]    11) Address: https://cagrid-
workflow.nci.nih.gov:8443/wsrf/services/cagrid/WorkflowFactoryService
[java]    12) Address:
https://10.120.79.112:8443/wsrf/services/cagrid/Dorian
[java]    13) Address: https://localhost:443/wsrf/services/cagrid/CaTIES
[java]    14) Address:
http://140.254.80.50:50100/wsrf/services/cagrid/OrderQueueService
[java]    15) Address:
https://137.187.66.137:58091/wsrf/services/cagrid/CaTissueCore
[java]    16) Address: https://cagrid-
workflow.nci.nih.gov:8443/wsrf/services/cagrid/FederatedQueryProcessor
[java]    17) Address: https://cagrid-gts-
slave.nci.nih.gov:8443/wsrf/services/cagrid/GTS
[java]    18) Address: https://cagrid-
cccfu.medctr.ad.wfubmc.edu:8443/wsrf/services/cagrid/GTS
[java]    19) Address:
http://127.0.0.1:18080/wsrf/services/cagrid/CaArraySvc
[java]    20) Address:
http://137.187.66.137:18080/wsrf/services/cagrid/CaArraySvc
[java]    21) Address:
https://192.168.200.198:8443/wsrf/services/cagrid/RegistrationConsuserService
[java]    22) Address:
https://localhost:8443/wsrf/services/cagrid/CaGridTutorialService
[java]    23) Address: https://cagrid-gts-
master.nci.nih.gov:8443/wsrf/services/cagrid/GTS
[java]    24) Address:
http://141.161.25.20:8080/wsrf/services/cagrid/GridPIR
[java]    25) Address: https://10.10.10.2:8443/wsrf/services/cagrid/Caaers
[java]    26) Address: https://10.0.2.13:8443/wsrf/services/cagrid/Dorian
[java]    27) Address: http://cacti-g5-12.wustl.edu:8080/wsrf-
canano/services/cagrid/CaNanoLabSvc
[java]    28) Address:
https://127.0.1.1:8443/wsrf/services/cagrid/HelloWorld
[java]    29) Address:
http://137.187.182.58:18080/wsrf/services/cagrid/CaArraySvc
[java]    30) Address:
https://caaers.mayo.edu:8443/wsrf/services/cagrid/CaaersDataService
[java]    31) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/GlobalModelExchange
[java]    32) Address:
http://cpas.nci.nih.gov:8080/wsrf/services/cagrid/CpasSvc

```

```

[java]      33) Address:
http://137.187.66.137:44210/wsrf/services/cagrid/Ctods
[java]      34) Address: https://192.168.88.79:38443/wsrf/services/cagrid/GTS
[java]      35) Address:
https://137.187.182.60:47210/wsrf/services/cagrid/CaTissueCore
[java]      36) Address: http://sonicserver-
dev.northwestern.edu:9000/wsrf/services/cagrid/HelloWorld
[java]      37) Address:
http://140.254.80.50:50101/wsrf/services/cagrid/ImageDataService
[java]      38) Address:
http://128.252.178.222:8080/wsrf/services/cagrid/GeneConnect
[java]      39) Address: http://cananolab.nci.nih.gov:80/wsrf-
canano/services/cagrid/CaNanoLabSvc
[java]      40) Address: http://cananogrid.bme.gatech.edu:8088/wsrf-
canano/services/cagrid/CaNanoLabSvc
[java]      41) Address:
http://127.0.0.1:8080/inp2/services/cagrid/InPxService
[java]      42) Address:
http://140.254.80.50:50100/wsrf/services/cagrid/DICOMDataService
[java]      43) Address:
http://140.254.80.50:50101/wsrf/services/cagrid/DICOMDataService
[java]      44) Address:
http://137.187.182.173:8080/wsrf/services/cagrid/NCIAQueryService
[java]      45) Address: https://cagrid-dorian-
qa.nci.nih.gov:8443/wsrf/services/cagrid/Dorian
[java]      46) Address:
https://152.11.189.189:8443/wsrf/services/cagrid/Caaers
[java]      47) Address: http://cabio-gridservice.nci.nih.gov:80/wsrf-
cabio/services/cagrid/CaBIOSvc
[java]      48) Address: https://cagrid-
auth.nci.nih.gov:8443/wsrf/services/cagrid/AuthenticationService
[java]      49) Address: https://192.168.88.79:8443/psc-
wsrf/services/cagrid/AdverseEventConsumer
[java]      50) Address: http://cananolab.abcc.ncifcrf.gov:8080/wsrf-
canano/services/cagrid/CaNanoLabSvc
[java]      51) Address: https://localhost:443/wsrf-
security/services/cagrid/Dorian
[java]      52) Address:
https://157.229.220.121:8443/wsrf/services/cagrid/Dorian
[java]      53) Address:
https://tropixdev04.mayo.edu:8443/wsrf/services/cagrid/AuthenticationService
[java]      54) Address:
http://137.187.182.60:42210/wsrf/services/cagrid/CAGWAS
[java]      55) Address: https://localhost:443/wsrf-
security/services/cagrid/AuthenticationService
[java]      56) Address: https://10.20.2.95:8443/wsrf/services/cagrid/Caaers
[java]      57) Address:
https://137.187.182.58:58091/wsrf/services/cagrid/CaTissueCore
[java]      58) Address:
http://cgwb.nci.nih.gov:80/wsrf/services/cagrid/CGWBDataService
[java]      59) Address:
https://128.101.190.104:8443/wsrf/services/cagrid/Sequest
[java]      60) Address: https://cagrid-
gridgrouper.nci.nih.gov:8443/wsrf/services/cagrid/GridGrouper
[java]      61) Address:
http://140.254.80.50:50010/wsrf/services/cagrid/PathologyDataService
[java]      62) Address:
http://cagrid.jax.org:18080/wsrf/services/cagrid/CaArraySvc
[java]      63) Address: http://caintegrator-
stage.nci.nih.gov:8080/wsrf/services/cagrid/CgomCgems
[java]      64) Address:
http://array.nci.nih.gov:80/wsrf/services/cagrid/CaArraySvc

```

```

[java]      65) Address:
http://140.254.80.84:8080/wsrf/services/cagrid/TMADataService
[java]      66) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java]      67) Address:
https://dorian.training.cagrid.org:6443/wsrf/services/cagrid/Dorian
[java]      68) Address:
https://157.229.221.119:8443/wsrf/services/cagrid/Dorian
[java]      69) Address: https://cagrid-
ccwfmu.medctr.ad.wfubmc.edu:8443/wsrf/services/cagrid/HelloWorld
[java]      70) Address: https://147.140.9.23:8443/wsrf/services/cagrid/Dorian
[java]      71) Address: https://dorian.bmi.ohio-
state.edu:9443/wsrf/services/cagrid/Dorian
[java]      72) Address:
https://tropixdev01.mayo.edu:8443/wsrf/services/cagrid/LexBIGCaGridServices
[java] ----

[java] =====
[java] Querying by Search String [Scott]
[java] =====
[java]      0) Address: https://cagrid-
workflow.nci.nih.gov:8443/wsrf/services/cagrid/FederatedQueryProcessor
[java]      1) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] ----

[java] =====
[java] Querying by Research Center Name [OSU]
[java] =====
[java] no results.
[java] ----

[java] =====
[java] Querying by POC
[gov.nih.nci.cagrid.metadata.common.PointOfContact@94f965d]
[java] =====
[java]      0) Address: https://cagrid-
workflow.nci.nih.gov:8443/wsrf/services/cagrid/FederatedQueryProcessor
[java]      1) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] ----

[java] =====
[java] Querying by Service name [CaDSRService]
[java] =====
[java]      0) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] ----

[java] =====
[java] Querying by Operation name [findAllProjects]
[java] =====
[java]      0) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] ----

[java] =====

```

```
[java] Querying by Operation input
[gov.nih.nci.cagrid.metadata.common.UMLClass@50c8e2fa]
=====
[java] 0) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] -----


[java] =====
[java] Querying by Operation output
[gov.nih.nci.cagrid.metadata.common.UMLClass@50c8e2fa]
=====
[java] 0) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] -----


[java] =====
[java] Querying by Operation class
[gov.nih.nci.cagrid.metadata.common.UMLClass@50c8e2fa]
=====
[java] 0) Address: http://cagrid-
service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService
[java] -----


BUILD SUCCESSFUL
Total time: 19 seconds
```

Using Ivy to download libraries from the remote caGrid repository

In the second example, we will configure Ivy to copy caGrid dependencies from a remote Ivy repository accessible over the Internet. We will no longer require a local caGrid installation or environment variable as was necessary for the previous example. As mentioned before, the approach we will be taking for integrating Ivy is to create another Ant build file, which extends our project's regular build process. We will be creating an alternative (`remove-ivy-build.xml`) to the `local-ivy-build.xml` used in the previous example. We will use all the same Ant targets names and processes however. Actually, the only difference in the local and remote examples, is the way in which we tell Ivy about the settings it should use. Whereas in the last example we pointed to settings in the caGrid release, in this example we will point to settings provided by the remote repository itself; all we need to know is the URL of these settings.

The build file is shown here:

```
<project name="ivyexamples-discovery-remote-ivy" default="all"
xmlns:ivy="antlib:org.apache.ivy.ant">

<import file="build.xml" />

<taskdef resource="org/apache/ivy/ant/antlib.xml"
uri="antlib:org.apache.ivy.ant">
    <classpath>
        <fileset dir="${lib.dir}" includes="ivy-*.*jar" />
    </classpath>
</taskdef>

<property name="ivy.settings.url"
value="http://software.cagrid.org/repository-1.2/ivysettings.xml" />
```

```

<ivy:settings id="ivy.instance" url="${ivy.settings.url}" override="true" />

<target name="ivy-resolve" description="--> retreive dependencies with ivy">
    <ivy:retrieve type="jar" />
</target>

<target name="ivy-report" depends="ivy-resolve" description="--> generates a report of dependencies">
    <ivy:report todir="${build.dir}" />
</target>

<target name="ivy-clean" description="--> cleans all the libraries which were retrieved for the project">
    <delete includeemptydirs="true" failonerror="false">
        <fileset dir="${lib.dir}">
            <include name="**/*" />
            <exclude name="ivy*.jar" />
        </fileset>
    </delete>
    <ivy:cleancache />
</target>

<target name="all" depends="ivy-clean,ivy-resolve,ivy-report" />
</project>

```

To actually run this process and copy the dependencies we need into our *lib* directory, we just need to type:

```
ant -f remote-ivy-build.xml
```

An example output of this can be see below:

```
Buildfile: remote-ivy-build.xml
[ivy:settings] :: Ivy 2.0.0-beta2 - 20080225093827 :: http://ant.apache.org/ivy/ ::
[ivy:settings] :: loading settings :: url =
http://software.cagrid.org/repository-1.2/ivysettings.xml

ivy-clean:

ivy-resolve:
[ivy:retrieve] :: resolving dependencies :: org.cagrid.ivyexamples#discovery;working@gcCity
[ivy:retrieve]   confs: [default]
[ivy:retrieve]     found caGrid#discovery;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found caGrid#core;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found castor#castor;0.9.9 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found jdom#jdom;1.0 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found caGrid#metadatautils;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found caGrid#metadata;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found jasciidammit#jasciidammit;1.1 in remote-caGrid-ivy-resolver
[ivy:retrieve]     found clarkware#jdepend;2.7 in remote-caGrid-ivy-resolver
[ivy:retrieve]     downloading http://software.cagrid.org/repository-1.2/caGrid/discovery/1.2/caGrid-discovery-tests-1.2.jar ...
[ivy:retrieve]       .... (13kB)
[ivy:retrieve]     ... (0kB)
[ivy:retrieve]     [SUCCESSFUL ] caGrid#discovery;1.2!caGrid-discovery-tests.jar (934ms)
[ivy:retrieve]     downloading http://software.cagrid.org/repository-1.2/caGrid/discovery/1.2/caGrid-discovery-1.2.jar ...
[ivy:retrieve]       .... (12kB)
[ivy:retrieve]     ... (0kB)
[ivy:retrieve]     [SUCCESSFUL ] caGrid#discovery;1.2!caGrid-discovery.jar (897ms)
[ivy:retrieve]     downloading http://software.cagrid.org/repository-1.2/caGrid/core/1.2/caGrid-core-tests-1.2.jar ...
[ivy:retrieve]       .... (16kB)
```

```
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#core;1.2!caGrid-core-tests.jar (939ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/core/1.2/caGrid-core-1.2.jar ...
[ivy:retrieve] ..... (82kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#core;1.2!caGrid-core.jar (1274ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadatautils/1.2/caGrid-metadatautils-1.2.jar ...
[ivy:retrieve] ..... (37kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#metadatautils;1.2!caGrid-metadatautils.jar
(1082ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/castor/castor/0.9.9/castor-0.9.9.jar ...
[ivy:retrieve] .....
[ivy:retrieve] .....
[ivy:retrieve] .....
[ivy:retrieve] ..... (1865kB)
[ivy:retrieve] [SUCCESSFUL ] castor#castor;0.9.9!castor.jar (6632ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/jdom/jdom/1.0/jdom-1.0.jar ...
[ivy:retrieve] ..... (149kB)
[ivy:retrieve] [SUCCESSFUL ] jdom#jdom;1.0!jdom.jar (1387ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/cagrid/types/data/data-1.2.xsd ...
[ivy:retrieve] .. (3kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#metadata;1.2!cagrid/types/data/data.xsd
(748ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/caGrid-metadata-security-1.2.jar ...
[ivy:retrieve] ..... (27kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#metadata;1.2!caGrid-metadata-security.jar
(1054ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/cagrid/types/caGridMetadata-1.2.xsd ...
[ivy:retrieve] .. (1kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ]
caGrid#metadata;1.2!cagrid/types/caGridMetadata.xsd (704ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/caGrid-metadata-data-1.2.jar ...
[ivy:retrieve] ..... (36kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#metadata;1.2!caGrid-metadata-data.jar
(1054ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/cagrid/types/security/security-1.2.xsd ...
[ivy:retrieve] .. (2kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ]
caGrid#metadata;1.2!cagrid/types/security/security.xsd (705ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/caGrid-metadata-common-1.2.jar ...
[ivy:retrieve] ..... (105kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] caGrid#metadata;1.2!caGrid-metadata-common.jar
(1327ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/cagrid/types/common/common-1.2.xsd ...
[ivy:retrieve] .. (6kB)
```

```
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ]
caGrid#metadata;1.2!cagrid/types/common/common.xsd (783ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/caGrid/metadata/1.2/cagrid/types/service/servicemodel-1.2.xsd ...
[ivy:retrieve] .. (6kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ]
caGrid#metadata;1.2!cagrid/types/service/servicemodel.xsd (784ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/jasciidammit/jasciidammit/1.1/jasciidammit-1.1.jar ...
[ivy:retrieve] .. (6kB)
[ivy:retrieve] [SUCCESSFUL ] jasciidammit#jasciidammit;1.1!jasciidammit.jar
(659ms)
[ivy:retrieve] downloading http://software.cagrid.org/repository-
1.2/clarkware/jdepend/2.7/jdepend-2.7.jar ...
[ivy:retrieve] ..... (423kB)
[ivy:retrieve] [SUCCESSFUL ] clarkware#jdepend;2.7!jdepend.jar (2084ms)
[ivy:retrieve] :: resolution report :: resolve 5940ms :: artifacts dl 23089ms
-----
|      |     modules    || artifacts |
| conf | number| search|dwnlded|evicted|| number|dwnlded|
-----
| default | 8 | 8 | 8 | 0 || 17 | 17 |
-----
[ivy:retrieve] :: retrieving :: org.cagrid.ivyexamples#discovery
[ivy:retrieve] confs: [default]
[ivy:retrieve] 12 artifacts copied, 0 already retrieved (2777kB/54ms)

ivy-report:
[ivy:report] Processing C:\Users\oster\.ivy2\cache\org.cagrid.ivyexamples-
discovery-default.xml to D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build\org.cagrid.ivyexamples-discovery-
default.html
[ivy:report] Processing C:\Users\oster\.ivy2\cache\org.cagrid.ivyexamples-
discovery-default.xml to D:\projects\caBIG\cagrid-1-
0\Demos\ivyexamples\discovery\build\org.cagrid.ivyexamples-discovery-
default.graphml

all:

BUILD SUCCESSFUL
Total time: 31 seconds
```

The output of this process is fairly similar to the previous example (twelve artifacts were still copied into our project), but note it actually took quite a bit longer to finish, as it was downloading files from the Internet. This leads to one trade off which should be looked at when determining which approach to take, as this process depends on the efficiency and availability of remote resources. Note however, that Ivy actually does cache these remote resources, and if we run `ivy-resolve` again (we can't just run our `all` target as that actually deletes the Ivy cache) it knows that it doesn't need to download them again. To try this, type:

```
ant -f remote-ivy-build.xml ivy-resolve
You should see output similar to this:
[oster@gcity ~...ivyexamples/discovery]$ ant -f remote-ivy-build.xml ivy-
resolve
Buildfile: remote-ivy-build.xml
[ivy:settings] :: Ivy 2.0.0-beta2 - 20080225093827 ::
http://ant.apache.org/ivy/ ::
```

```
[ivy:settings] :: loading settings :: url =
http://software.cagrid.org/repository-1.2/ivysettings.xml

ivy-resolve:
[ivy:retrieve] :: resolving dependencies :: 
org.cagrid.ivyexamples#discovery;working@gcity
[ivy:retrieve] confs: [default]
[ivy:retrieve] found caGrid#discovery;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve] found caGrid#core;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve] found castor#castor;0.9.9 in remote-caGrid-ivy-resolver
[ivy:retrieve] found jdom#jdom;1.0 in remote-caGrid-ivy-resolver
[ivy:retrieve] found caGrid#metadatautils;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve] found caGrid#metadata;1.2 in remote-caGrid-ivy-resolver
[ivy:retrieve] found jasciidammit#jasciidammit;1.1 in remote-caGrid-ivy-
resolver
[ivy:retrieve] found clarkware#jdepend;2.7 in remote-caGrid-ivy-resolver
[ivy:retrieve] :: resolution report :: resolve 375ms :: artifacts dl 23ms
-----
|      |      modules      ||  artifacts  |
|  conf  |  number| search|dwnlded|evicted||  number|dwnlded|
-----
|  default  |  8  |  0  |  0  |  0  ||  17  |  0  |
-----
[ivy:retrieve] :: retrieving :: org.cagrid.ivyexamples#discovery
[ivy:retrieve] confs: [default]
[ivy:retrieve] 0 artifacts copied, 12 already retrieved (0kB/12ms)

BUILD SUCCESSFUL
Total time: 1 second
```

Note this time it was significantly faster, and that it told us it didn't need to retrieve anything new. You can experiment with this, by removing a few of the jars from the `lib` directory, and doing another `ivy-resolve` and you'll see it only downloads what it needs.

You can now run the main build file just as before, and you should see the same results.

Looking at Ivy reports as a guide to manually copying libraries from a local caGrid installation

The preceding two examples showed how Ivy could be used programmatically in your build process to copy dependencies into your project. If for some reason you do not have the ability to integrate with Ivy in this way, you can use the output of the Ivy reports Ivy can generate (as shown in the preceding examples), to look at any given project's dependencies, and manually copy the dependencies you need (though it is recommended you allow Ivy to do this for you, even if you don't do it as part of your build process).

Shown below is a portion of the report generated by running the `ivy-report` target of the ivy builds files:

discovery by org.cagrid.ivyexamples

resolved on 2008-03-12 23:50:58



default

Dependencies Stats	
Modules	8
Revisions	8 (8 searched 0%, 0 downloaded 0%, 0 evicted 0%, 0 errors 0%)
Artifacts	17 (0 downloaded, 0 failed)
Artifacts size	2799 kB (0 kB downloaded, 2799 kB in cache)

Dependencies Overview

Module	Revision	Status	Resolver	Default	Licenses	Size
discovery by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	26 kB
--- metadatautils by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	38 kB
----- jasciidammit by jasciidammit	1.1	integration	caGrid-fs-fs	false	LGPL	7 kB
----- metadata by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	191 kB
----- core by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	99 kB
----- jdepend by clarkware	2.7	integration	caGrid-fs-fs	false		424 kB
----- jdom by jdom	1.0	release	caGrid-fs-fs	false		150 kB
----- castor by castor	0.9.9	integration	caGrid-fs-fs	false	Apache	1865 kB
--- metadata by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	191 kB
--- core by caGrid	1.2	integration	caGrid-local-fs-fs	false	caGrid	99 kB
----- jdepend by clarkware	2.7	integration	caGrid-fs-fs	false		424 kB
----- jdom by jdom	1.0	release	caGrid-fs-fs	false		150 kB
----- castor by castor	0.9.9	integration	caGrid-fs-fs	false	Apache	1865 kB

You can see in the Dependencies Overview section, a tree-like representation of the transitive dependencies are shown. In our example, the example project only depends on caGrid Discovery, which in turn depends on the metadatautils, metadata, and core caGrid projects. These projects in turn have dependencies which are also retrieved. You can use this information to understand which libraries you may need for a given use of any caGrid project, and why you need what you need. The full report contains the specific details of each module's artifacts. You can find Ivy reports for every caGrid project in each project's `ext` directory.

Ivy Summary

This section has shown a variety of ways in which the power of an Ivy-enabled caGrid build can be leveraged. However, it has only scratched the surface of what Ivy can do, and those who find utility in the mechanisms described here are encouraged to look further into Ivy. Those who are doing extensive development work depending on caGrid will likely find it a very convenient way to stay integrated with both released versions of caGrid, as well as supporting nightly or continuous integration versions; switching between the two can be as simple as pointing to a different Ivy settings file. It also ensures your project will be more "future proof" in that when new caGrid releases are made available, you can automatically update your projects to leverage them.

Chapter 3 caGrid Metadata Infrastructure

This chapter describes the caGrid metadata infrastructure, the Discovery API, the caDSR Grid Service, and the EVS APIs.

Topics in this chapter include:

- Code Example Information on this page
- Metadata API Usage Overview on page 41
- Discovery API Usage Overview on page 49
- caDSR Grid Service Usage Overview on page 68
- EVS API Usage Overview on page 79

Code Example Information

Unless otherwise fully specified, the following set of imports can be assumed for the code examples provided in this chapter (they are omitted to make the examples more readable).

```
import gov.nih.nci.cadsr.umlproject.domain.*;
import gov.nih.nci.cagrid.cadsr.client.*;
import gov.nih.nci.cagrid.cadsrservice.*;
import gov.nih.nci.cagrid.discovery.client.*;
import gov.nih.nci.cagrid.metadata.*;
import gov.nih.nci.cagrid.metadata.common.*;
import gov.nih.nci.cagrid.metadata.dataservice.*;
import gov.nih.nci.cagrid.metadata.exceptions.*;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.apache.axis.types.URI.MalformedURIException;
```

Metadata API Usage Overview

The following link provides a reference to the technical architecture and design document(s) for caGrid metadata:

<http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/~checkout~/cagrid-1-0/Documentation/docs/metadata/caGrid-metadata-infrastructure-design.doc?rev=1.12;content-type=application%2Foctet-stream;cvsroot=cagrid-1-0>

All caGrid services are expected to expose a standard set of service metadata. Details about this design and the specifics of the metadata can be found in the caGrid Metadata Design Document. This section describes the high-level API, which can be used to access and manipulate instances of this metadata. All the standard metadata models are representations, which can be used to programmatically interact with the models. Figure 3-1 and Figure 3-2 are the Standard Service (`ServiceMetadata`) and Data Service (`DomainModel`) metadata models respectively. The APIs described here can be used access these models from services, and serialize and deserialize them to and from XML. These methods complement the Discovery API. Once an EPR (End Point Reference) is returned from the Discovery API, these methods can be used to access and inspect the full metadata.

Figure 3-1 Service Model

Figure 3-2 Data Service metadata model

The `ResourcePropertyHelper` API, not detailed here, is the lower level API, which can be used to directly gather information about `ResourceProperties` (this is how metadata is exposed in caGrid). The `MetadataUtils`, described here, leverage this API, and expose some of its exceptions. The possible exceptions generated by the metadata utility methods are detailed in Figure 3-3.

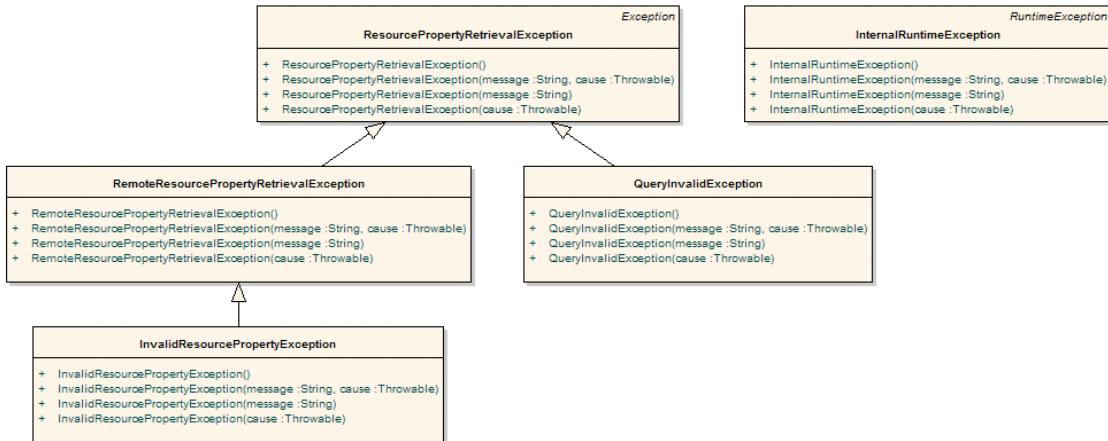


Figure 3-3 Metadata Exceptions

A non-discriminating client may simply opt to catch `ResourcePropertyRetrievalException`, as it is the base-checked exception. An additional non-checked exception, `InternalRuntimeException`, can also be thrown but is solely used to represent an internal logic error in the APIs. It is not expected clients can “recover” from such an exception. As such, clients should not attempt to catch this runtime exception for any other reason than to mask the problem.

`QueryInvalidException` is thrown if an invalid XPath query is issued. Problems originating from remote services are thrown in the subclass `RemoteResourcePropertyRetrievalException`. During general use of the metadata utilities, this is the most likely exception clients may see, as it is thrown if a service is not properly exposing the proper metadata. Clients leveraging the lower level resource property APIs should take care to appropriately address each type of exception if they are communicating with services. For example, even though it is a caBIG requirement to expose the standard service metadata, clients should properly handle the case where it is not present. Asking for specific metadata that a service does not provide would yield an `InvalidResourcePropertyException`.

API Details

`gov.nih.nci.cagrid.metadata.MetadataUtils`

Member Function Documentation

static ServiceMetadata
`gov.nih.nci.cagrid.metadata.MetadataUtils.getServiceMetadata(EndpointReferenceType serviceEPR)` throws `InvalidResourcePropertyException`, `RemoteResourcePropertyRetrievalException`, `ResourcePropertyRetrievalException` [static]

Obtain the service metadata from the specified service.

Parameters:

`serviceEPR`

Returns:

Exceptions:

InvalidResourcePropertyException
 RemoteResourcePropertyRetrievalException
 ResourcePropertyRetrievalException



static DomainModel gov.nih.nci.cagrid.metadata.MetadataUtils.getDomainModel(EndpointReferenceType serviceEPR) throws InvalidResourcePropertyException, RemoteResourcePropertyRetrievalException, ResourcePropertyRetrievalException [static]

Obtain the data service metadata from the specified service.

Parameters:

serviceEPR

Returns:**Exceptions:**

InvalidResourcePropertyException
 RemoteResourcePropertyRetrievalException
 ResourcePropertyRetrievalException



static void gov.nih.nci.cagrid.metadata.MetadataUtils.serializeServiceMetadata(ServiceMetadata metadata, Writer writer) throws Exception [static]

Write the XML representation of the specified metadata to the specified writer. If either is null, an IllegalArgumentException will be thrown.

Parameters:

metadata
 writer

Exceptions:

Exception

static ServiceMetadata gov.nih.nci.cagrid.metadata.MetadataUtils.deserializeServiceMetadata(Reader xmlReader) throws Exception [static]

Create an instance of the service metadata from the specified reader. The reader must point to a stream that contains an XML representation of the metadata. If the reader is null, an IllegalArgumentException will be thrown.

Parameters:

xmlReader

Returns:**Exceptions:**

Exception



static void gov.nih.nci.cagrid.metadata.MetadataUtils.serializeDomainModel (DomainModel domainModel, Writer writer) throws Exception [static]

Write the XML representation of the specified metadata to the specified writer. If either is null, an IllegalArgumentException will be thrown.

Parameters:

domainModel
writer

Exceptions:

Exception

static DomainModel gov.nih.nci.cagrid.metadata.MetadataUtils.deserializeDomainModel (Reader xmlReader) throws Exception [static]

Create an instance of the data service metadata from the specified reader. The reader must point to a stream that contains an XML representation of the metadata. If the reader is null, an IllegalArgumentException will be thrown.

Parameters:

xmlReader

Returns:

Exceptions:

Exception



API Usage Examples

This section describes typical usage of the Metadata API. The exception handling shown in the code examples is not recommended practice, and is simplistic for demonstration purposes. The `MetadataUtils` class is the primary means of accessing and manipulating service metadata. It provides a number of static utility methods that can be directly invoked. This API provides an abstraction layer over lower-level APIs, specializing them to deal with the standard metadata types. Clients wishing to work with custom (or non-standard) metadata need to use the lower-level APIs and can consult the source code of the `MetadataUtils` class for guidance.

Accessing Metadata from a Service

In order to access a service's metadata, an End Point Reference pointing to the service must be provided. This can be obtained as a direct result of an invocation of a discovery method from the Discovery API, or manually constructed by specifying the service's *Address*. Examples of both can be found in the [Discovery API Usage Overview](#) starting on page 49.

As caBIG requires that standard metadata be made publicly available, client credentials are not necessary for invocation of these methods.

The first example, shown in Figure 3-4, demonstrates accessing a service's standard `ServiceMetadata`, which is common to all caGrid services. As described above, the first step is to obtain an appropriate EPR (line 1). Given this EPR, the `MetadataUtils`'s `getServiceMetadata` method, shown on line 4 in Figure 3-4, can be used to obtain the bean representation of the metadata. Upon successful completion of this method, the fully populated bean can be inspected to obtain the information of interest. Several exceptions, subclassed from the base `ResourcePropertyRetrievalException`, can be thrown by this operation. A non-discriminating client may choose to simply handle this base exception. Additional details on the other exceptions, and why they may be thrown, are described in the [Metadata API Usage Overview](#) on page 41, as well as the javadoc of the APIs.

```

1   EndpointReferenceType serviceEPR = /* Some service's EPR */
2   try {
3       ServiceMetadata serviceMetadata =
4           MetadataUtils.getServiceMetadata(serviceEPR);
5   } catch (InvalidResourcePropertyException e) {
6       // TODO handle the case where the service doesn't expose
7       // standard metadata
8       e.printStackTrace();
9   } catch (RemoteResourcePropertyRetrievalException e) {
10      // TODO handle the case where there was some other problem
11      // communicating with the service (unavailable, untrusted, etc)
12      e.printStackTrace();
13 } catch (ResourcePropertyRetrievalException e) {
14     // TODO handle all other general error (such as inability to
15     // deserialize the result)
16     e.printStackTrace();
17 }
```

Figure 3-4 Accessing standard service metadata

The process for accessing data service `DomainModel` metadata, shown in Figure 3-5, is the same as accessing standard metadata. Once the metadata is obtained, in line 3, it can be inspected, as shown in line 4 where the long name of the project being exposed by the data service is printed to the console.

```

1   EndpointReferenceType serviceEPR = /* Some service's EPR */
2   try {
3       DomainModel domainModel = MetadataUtils.getDomainModel(serviceEPR);
4       System.out.println(domainModel.getProjectLongName());
5   } catch (InvalidResourcePropertyException e) {
6       // TODO handle the case where the service doesn't expose
7       // standard data service metadata
8       e.printStackTrace();
9   } catch (RemoteResourcePropertyRetrievalException e) {
10      // TODO handle the case where there was some other problem
11      // communicating with the service (unavailable, untrusted, etc)
12      e.printStackTrace();
13 } catch (ResourcePropertyRetrievalException e) {
14     // TODO handle all other general error (such as inability to
15     // deserialize the result)
16     e.printStackTrace();
```

Figure 3-5 Accessing standard data service metadata

Processing Metadata as XML

In addition to accessing metadata from services, the `MetadataUtils` provide the capability to read and write metadata instances as XML documents. This can be useful for not only storage and display of metadata, but also for exposing grid service metadata as XML. These methods also provide a way to inspect the metadata in object (bean) form.

In Figure 3-6, example code is shown that saves an instance of standard service metadata to a file named `serviceMetadata.xml`. The metadata instance, defined in line 1, can be acquired using code similar to that shown in Figure 3-6, or by some other mechanism. The `serializeServiceMetadata` method can be then passed this instance, and an instance of the `java.io.Writer` interface, as shown on line 11. Any Writer implementation works, but the example below shows using a `FileWriter`, on line 5, to write the metadata to the specified file. After the `MetadataUtils` have been used to write the metadata to XML, the Writer used should be closed, as shown on line 18. Though not shown, a similar method, `serializeDomainModel`, exists for writing data service metadata to XML; its usage pattern is the same.

```
1  ServiceMetadata serviceMetadata = /* Some service's Metadata */  
2  Writer writer = null;  
3  try {  
4      //Any Writer implementation will work  
5      writer = new FileWriter("serviceMetadata.xml");  
6  } catch (IOException e) {  
7      // TODO Handle problem setting up writer  
8      e.printStackTrace();  
9  }  
10 try {  
11     MetadataUtils.serializeServiceMetadata(serviceMetadata, writer);  
12 } catch (Exception e) {  
13     // TODO handle problem serializing to the specified writer  
14     e.printStackTrace();  
15 }  
16 try {  
17     //be sure to close your Writer  
18     writer.close();  
19 } catch (IOException e) {  
20     // TODO handle problem closing the writer  
21     e.printStackTrace();  
22 }
```

Figure 3-6 Serializing metadata to a file

As a complement to the serialization methods described and shown above, deserialization methods also exist which read XML representations of metadata and return appropriately populated metadata beans. In Figure 3-7, example code is shown which populates a new `ServiceMetadata` instance from an XML representation stored in a file named `serviceMetadata.xml`. This code, used in conjunction with the previous example, reconstitutes the original metadata instance. Similar to the serialization methods that use a `java.io.Writer`, the deserialization methods use a `java.io.Reader` to read the XML representation. In the example below, a `FileReader` is used on line 4. This `Reader` is then passed to the `deserializeServiceMetadata` method on line 11, and

the populated `ServiceMetadata` instance is returned. As with the Writer instance in the serialization methods, the Reader instance should be closed once it is used (as shown on line 18). Though not shown, a similar method, `deserializeDomainModel`, exists for reading data service metadata from XML; its usage pattern is the same.

```

1   Reader reader = null;
2   try {
3       // Any Reader will work
4       reader = new FileReader("serviceMetadata.xml");
5   } catch (FileNotFoundException e) {
6       // TODO handle the case where the file is not found
7       e.printStackTrace();
8   }
9   try {
10       ServiceMetadata serviceMetadata =
11           MetadataUtils.deserializeServiceMetadata(reader);
12   } catch (Exception e) {
13       // TODO handle problem deserializing from the reader
14       e.printStackTrace();
15   }
16   try {
17       //be sure to close your reader
18       reader.close();
19   } catch (IOException e) {
20       // TODO handle problem closing the reader
21       e.printStackTrace();
22   }

```

Figure 3-7 Deserializing metadata from a file

Discovery API Usage Overview

The Discovery API provides an abstraction over the standard operations used to query the Index Service. It provides a number of operations that can be used to discover services of interest. The basic process of use is to construct an instance of the `DiscoveryClient`, optionally specifying the End Point Reference (EPR) of the Index Service to query, and then invoking the appropriate discovery methods. Each method returns an array of EPRs of the matching appropriate services. These returned EPRs can then be used to invoke the services, or ask them for their metadata for further discrimination. It is worth noting that the Index Service, as an aggregated source of distributed information, inherently operates on out of date information. It is possible that services that are running do not yet have their metadata aggregated in the Index Service, and it is possible that services present in the Index Service have recently been taken down. caGrid attempts to strike a balance between performance and reliability of information in the Index Service. The information returned by the Discovery API should be accurate within a few minutes, but applications building upon it should be aware of this, and should not assume a service in the Index Service will always be available when it is invoked.

The `DiscoveryClient` uses the lower level “metadata utils” project to communicate with the Index Service. It exposes the exceptions generated from this lower level API, instead

of wrapping them with discovery-specific exceptions. The possible exceptions that discovery methods can throw are detailed below in Figure 3-8. A non-discriminating client may simply opt to catch `ResourcePropertyRetrievalException`, as it is the base checked exception. An additional non-checked exception, `InternalRuntimeException`, can also be thrown, but it is solely used to represent an internal logic error in the APIs and so it is not expected clients can “recover” from such an exception. As such, clients should not attempt to catch this runtime exception for any other reason than to mask the problem. Generic problems caused by the `DiscoveryClient` itself are thrown in the base

`ResourcePropertyRetrievalException`. A subclass of it, `QueryInvalidException`, is thrown if an invalid XPath query is issued. Unless the `DiscoveryClient` is extended, it is not expected that clients should encounter this. Problems originating from remote services are thrown in the subclass, `RemoteResourcePropertyRetrievalException`. During general use of the metadata utilities, this is the most likely exception clients may see, as it is thrown if a service is not properly exposing the proper metadata. In the context of the `DiscoveryClient`, it is not expected clients should experience any exceptions unless there is an issue with the Index Service. However, clients leveraging the lower level APIs should take more care to appropriately address each type of exception if they are communicating with other (community provided) services. For example, even though it is a caBIG requirement to expose the standard service metadata, clients should properly handle the case where it is not present. Asking for specific metadata that a service does not provide would yield an `InvalidResourcePropertyException`.

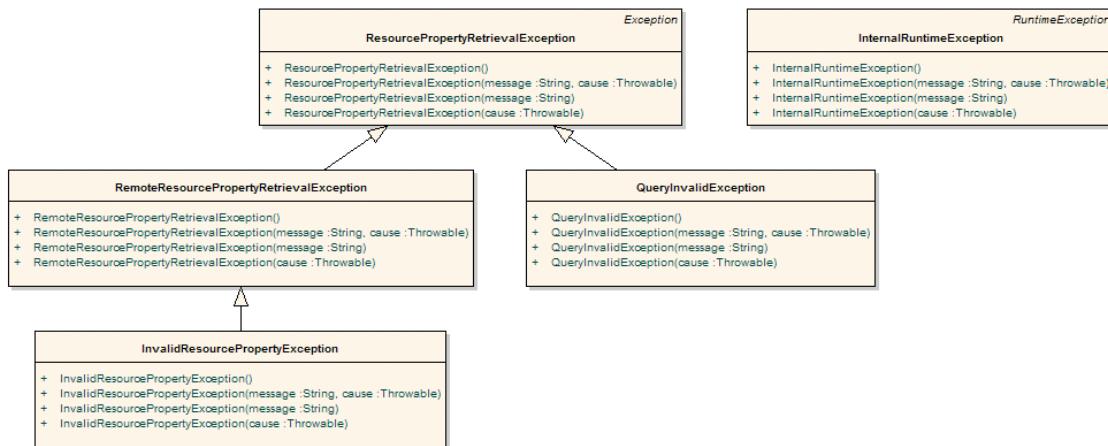


Figure 3-8 Metadata exceptions

While the methods in the API are designed around the caGrid standard metadata, it is also acceptable to have services register additional domain or application specific metadata to the Index Service. The Discovery API is designed for easy extensibility, such that additional application or domain specific discovery scenarios can be provided to compliment such additional metadata. The “business logic” of the `DiscoveryClient`, consists almost entirely of constructing appropriate XPath queries over the appropriate metadata, and leveraging lower-level APIs to actually invoke the queries. These lower-level APIs are made available to extenders of the client, such that they need only construct appropriate XPath queries to implement additional discovery scenarios.

The `DiscoveryClient` uses commons-logging to log general and debugging information. If configured to DEBUG level, the client prints out the XPaths it is sending to the Index Service, which may facilitate the creation of new discovery operations, or help track down problems.

As with most Globus clients, a properly configured `client-config.wsdd` file must be accessible by the underlying Axis engine. The simplest way to do this is to either run with your `$GLOBUS_LOCATION` as the “working directory,” add `$GLOBUS_LOCATION` to your classpath, or copy `$GLOBUS_LOCATION/client-config.wsdd` to your working directory or classpath. If you don’t do this, you will most likely see an exception similar to that shown in Figure 3-9, when you run the `DiscoveryClient`.

```
gov.nih.nci.cagrid.metadata.exceptions.RemoteResourcePropertyRetrievalException
: org.xml.sax.SAXException:SimpleDeserializer encountered a child element,
which is NOT expected, in something it was trying to deserialize.
```

Figure 3-9 Common `DiscoveryClient` exception

API Details

`gov.nih.nci.cagrid.discovery.client.DiscoveryClient`

`DiscoveryClient` represents the base discovery API. The client should be constructed passing a URL of an Index Service. Services can then be discovered by calling the `discover` methods and passing in the necessary criteria. The methods all return an `EndPointReferenceType[]`. See the main method for examples. This should be extended to provide specialized service-type discovery (beyond data services).

Constructor Documentation

`gov.nih.nci.cagrid.discovery.client.DiscoveryClient`.`DiscoveryClient () throws MalformedURIException`

Uses the Default Index Service.

Exceptions:

`MalformedURIException` if the Default Index Service is invalid.

```
DiscoveryClient.java.gov.nih.nci.cagrid.discovery.client.DiscoveryClient.Discov
eryClient (EndpointReferenceType indexEPR)
```

Uses the specified Index Service.

Parameters:

`indexEPR` the EPR to the Index Service to use.

Member Function Documentation

`EndpointReferenceType []`

`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getAllServices (boolean requireMetadataCompliance) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException`

Query the registry for all registered services.

Parameters:

`requireMetadataCompliance` if true, only services providing the standard metadata will be returned. Otherwise, all services registered will be returned, regardless of whether or not any metadata has been aggregated.

Returns:

`EndpointReferenceType[]` contains all registered services.

Exceptions:

`ResourcePropertyRetrievalException`
`QueryInvalidException`
`RemoteResourcePropertyRetrievalException`

`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getAllServices` → `gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter`

`EndpointReferenceType[]`
`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesBySearchString (String searchString)` throws `RemoteResourcePropertyRetrievalException`,
`QueryInvalidException`, `ResourcePropertyRetrievalException`

Searches ALL metadata to find occurrence of the given string. The search string is case-sensitive.

Parameters:

`searchString` the search string.

Returns:

`EndpointReferenceType[]` matching the criteria

Exceptions:

`ResourcePropertyRetrievalException`
`QueryInvalidException`
`RemoteResourcePropertyRetrievalException`

`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesBySearchString` → `gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter`

`EndpointReferenceType[]`
`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByResearchCenter (String centerName)` throws `RemoteResourcePropertyRetrievalException`,
`QueryInvalidException`, `ResourcePropertyRetrievalException`

Searches research center info to find services provided by a given cancer center.

Parameters:

`centerName` research center name

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
 QueryInvalidException
 RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByPointOfContact (PointOfContact *contact*) throws
 RemoteResourcePropertyRetrievalException, QueryInvalidException,
 ResourcePropertyRetrievalException

Searches to find services that have the given point of contact associated with them. Any fields set on the point of contact are checked for a match. For example, you can set only the lastName, and only it will be checked, or you can specify several fields and they all must be equal.

Parameters:

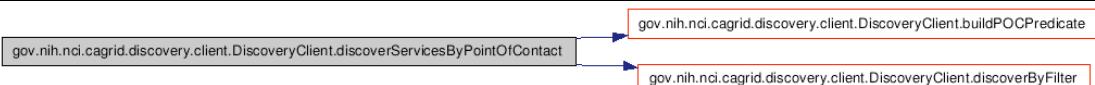
contact point of contact

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
 QueryInvalidException
 RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByName (String *serviceName*) throws RemoteResourcePropertyRetrievalException,
 QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have a given name.

Parameters:

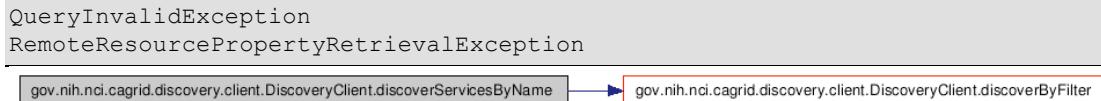
serviceName The service's name

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

ResourcePropertyRetrievalException



`EndpointReferenceType []`
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.`discoverServicesByConceptCode` (String *conceptCode*) throws `RemoteResourcePropertyRetrievalException`, `QueryInvalidException`, `ResourcePropertyRetrievalException`

Searches to find services based on the given concept code.

Parameters:

`conceptCode` A concept code the service is based on.

Returns:

`EndpointReferenceType []` matching the criteria

Exceptions:

`ResourcePropertyRetrievalException`
`QueryInvalidException`
`RemoteResourcePropertyRetrievalException`



`EndpointReferenceType []`
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.`discoverServicesByOperationName` (String *operationName*) throws `RemoteResourcePropertyRetrievalException`, `QueryInvalidException`, `ResourcePropertyRetrievalException`

Searches to find services that have a given operation.

Parameters:

`operationName` The operation's name

Returns:

`EndpointReferenceType []` matching the criteria

Exceptions:

`ResourcePropertyRetrievalException`
`QueryInvalidException`
`RemoteResourcePropertyRetrievalException`



`EndpointReferenceType []`
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.`discoverServicesByOperationInput` (UMLClass *clazzPrototype*) throws `RemoteResourcePropertyRetrievalException`, `QueryInvalidException`, `ResourcePropertyRetrievalException`

Searches to find services that have an operation defined that takes the given UMLClass as input. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

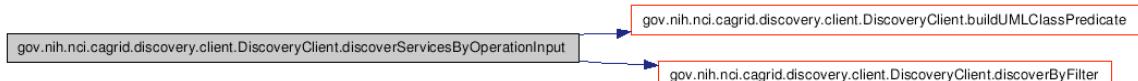
`clazzPrototype` The prototype UMLClass

Returns:

`EndpointReferenceType []` matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



`EndpointReferenceType []`
`gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperation`
Output (UMLClass `clazzPrototype`) throws
`RemoteResourcePropertyRetrievalException`, `QueryInvalidException`,
`ResourcePropertyRetrievalException`

Searches to find services that have an operation defined that produces the given UMLClass. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

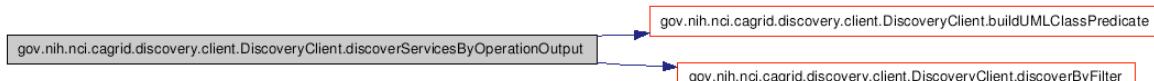
`clazzPrototype` The prototype UMLClass

Returns:

`EndpointReferenceType []` matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



EndpointReferenceType []
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperation
 Class (UMLClass *clazzPrototype*) throws
 RemoteResourcePropertyRetrievalException, QueryInvalidException,
 ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces the given UMLClass or takes it as input. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

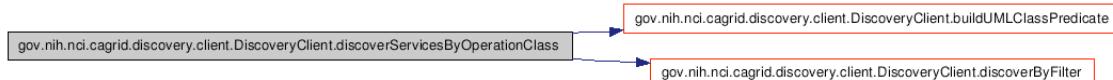
`clazzPrototype` The prototype UMLClass

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



EndpointReferenceType []
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperation
 ConceptCode (String *conceptCode*) throws
 RemoteResourcePropertyRetrievalException, QueryInvalidException,
 ResourcePropertyRetrievalException

Searches to find services that have an operation based on the given concept code

Parameters:

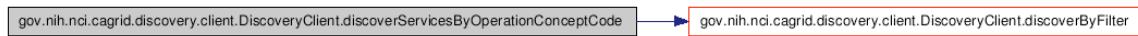
`conceptCode` The concept to look for

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByDataConceptCode (String *conceptCode*) throws
RemoteResourcePropertyRetrievalException, QueryInvalidException,
ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces the or takes as input, a Class with an attribute, attribute value domain , enumerated value meaning, or the class itself based on the given concept code.

Parameters:

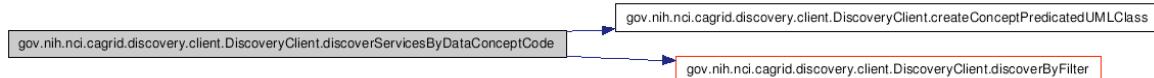
`conceptCode` The concept to look for

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByPermissibleValue (String *value*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produce or take as input, a Class with an attribute allowing the given value.

Parameters:

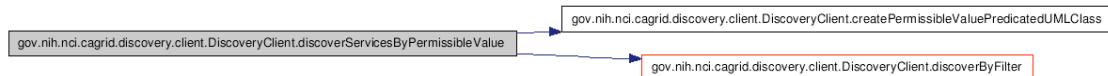
`value` The permissible value to look for

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```



String

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createPermissibleValuePredicatedUMLClass (String *value*, boolean *isDataService*) [protected]

Creates a UMLClass step that is predicated to contain either an attribute, attribute value domain, or enumerated value meaning, or the class itself based on that concept.

Parameters:

conceptCode the code to look for

Returns:

String

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createConceptPredicatedUMLClass (String *conceptCode*, boolean *isDataService*) [protected]

Creates a UMLClass step that is predicated to contain either an attribute, attribute value domain, enumerated value meaning, or the class itself based on that concept.

Parameters:

conceptCode the code to look for

Returns:

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getAllDataServices () throws
RemoteResourcePropertyRetrievalException, QueryInvalidException,
ResourcePropertyRetrievalException

Query the registry for all registered data services.

Returns:

EndpointReferenceType [] contain all registered services.

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByDomainModel (String *modelName*) throws
RemoteResourcePropertyRetrievalException, QueryInvalidException,
ResourcePropertyRetrievalException

Searches to find data services that are exposing a subset of given domain (short name or long name).

Parameters:

modelName The model to look for

Returns:

EndpointReferenceType [] matching the criteria

Exceptions:

```
ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException
```

```
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByDomainModel --> gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter
```

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByModelConceptCode (String *conceptCode*) throws
RemoteResourcePropertyRetrievalException, QueryInvalidException,
ResourcePropertyRetrievalException

Searches to find data services that expose a Class with an attribute , attribute value domain, enumerated value meaning, or the class itself based on the given concept code.

Parameters:

conceptCode The concept to look for

Returns:**Exceptions:**

```
RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException
```

```
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByModelConceptCode --> gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createConceptPredicatedUMLClass
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createConceptPredicatedUMLClass --> gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter
```

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByExposedClass (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches for data services that expose the given class. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. *UMLAttributeCollection* and *SemanticMetadataCollection*) are ignored).

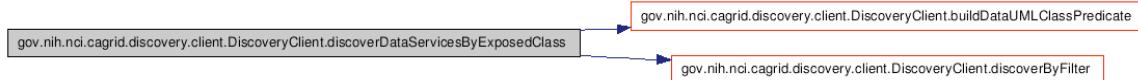
Parameters:

clazzPrototype The prototype UMLClass
clazzPrototype

Returns:

Exceptions:

```
RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException
```



EndpointReferenceType []
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByPermissibleValue (String *permissibleValue*) throws
 RemoteResourcePropertyRetrievalException, QueryInvalidException,
 ResourcePropertyRetrievalException

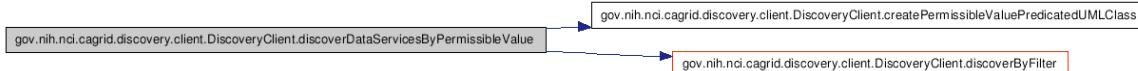
Searches for data services that expose a class with an attribute allowing the given value.

Parameters:

value The permissible value to look for

Returns:**Exceptions:**

```
RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException
```



EndpointReferenceType []
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByAssociationsWithClass (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException,
 QueryInvalidException, ResourcePropertyRetrievalException

Searches for data services that expose an association to or from the given class. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

clazzPrototype

Returns:

Exceptions:

```
RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException
```



static String
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildPOCPredicate
 (PointOfContact *contact*) [static, protected]

Builds up a predicate for a PointOfContact, based on the prototype passed in.

Parameters:

contact the prototype POC

Returns:

"" if the prototype has no non-null or non-whitespace values, or the predicate necessary to match all values.



static String
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildUMLClassPredicate
 (UMLClass *clazz*) [static, protected]

Builds up a predicate for a UMLClass, based on the prototype passed in.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

clazz the prototype UMLClass

Returns:

"" if the prototype has no non-null or non-whitespace values, or the predicate necessary to match all values.



static String
 gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildDataUMLClassPredicate
 (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazz*) [static, protected]



static String
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.addNonNullPredicate (String
name, String *value*, boolean *isAttribute*) [static, protected]

Parameters:

name the element or attribute name to check

value the value to add the predicate filter against if this is null or whitespace only, no predicated is added.

isAttribute whether or not name represents an attribute or element

Returns:

"" or the specified predicate (prefixed with " and ")

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter (String
xpathPredicate) throws RemoteResourcePropertyRetrievalException,
QueryInvalidException, ResourcePropertyRetrievalException [protected]

Applies the specified predicate to the common path in the Index Service's Resource Properties to return registered services' EPRs that match the predicate.

Parameters:

xpathPredicate predicate to apply to the "Entry" in Index Service

Returns:

EndpointReferenceType [] of matching services @

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.translateXPath (String
xpathPredicate) [protected]

Adds the common Index RP Entry filter, and translates the xpath to IndexService friendly XPath.

Parameters:

xpathPredicate

Returns:

the modified xpath

EndpointReferenceType

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getIndexEPR ()

Gets the EPR of the Index Service being used.

```
void gov.nih.nci.cagrid.discovery.client.DiscoveryClient.setIndexEPR
(EndpointReferenceType indexEPR)
```

Sets the EPR of the Index Service to use.

Parameters:

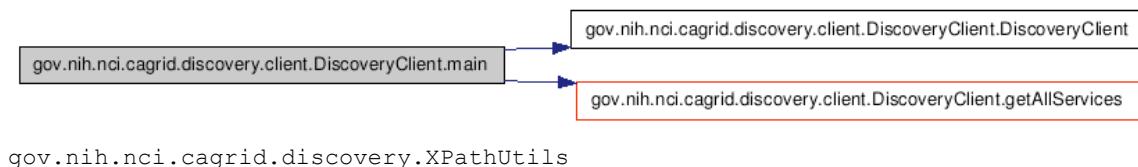
indexEPR the EPR of the Index Service to use.

```
static void gov.nih.nci.cagrid.discovery.client.DiscoveryClient.main (String[]
args) [static]
```

testing stub

Parameters:

args optional URL to Index Service to query.



Member Function Documentation

```
static String gov.nih.nci.cagrid.discovery.XPathUtils.translateXPath (String
prefixedXpath, Map namespaces) [static]
```

This utility takes an XPath that uses namespace prefixes (such as `/a:B/a:C`) and converts it to one without prefixes, by using the appropriate operators instead (such as `/*[namespace-uri()='http://DOMAIN.COM/SCHEMA' and local-name()='B']/*[namespace-uri()='http://DOMAIN.COM/SCHEMA' and local-name()='C']`). The only conceivable use for this function is to write sane XPath and convert it to the insane XPath Globus index service supports.

Note: This is not perfect. The known limitations are: 1) its overly aggressive, and will replace QName-looking string literals, 2) it will not work if you have namespaces attributes 3) it will silently not replace any QNames that you haven't supplied a prefix mapping for

Parameters:

`prefixedXpath` An xpath [optionally] using namespace prefixes in nodetests

`namespaces` A Map<String, String> keyed on namespace prefixes to resolve in the xpath, where the value is the actual namespace that should be used.

Returns:

a converted, conformant, xpath

API Usage Examples

This section describes typical usage of the Discovery API. The exception handling shown in the code examples is not recommended practice and is simplistic for demonstration purposes. Additional examples can be found in the source code of the

discovery project, in the main of the DiscoveryClient itself, as well as in the test source directory.

The main method of the DiscoveryClient can be run from the project's source folder by entering `ant runClient`. The discovery unit tests can also be run by entering `ant test`. The unit tests do not actually communicate with the Index Service; rather they simulate it with a Mock object.

Configuring an Index Service

The first step in using the Discovery API is constructing an instance of the DiscoveryClient. There are three constructors that can be used. The first, shown in line 7 of Figure 3-10, takes no arguments, and indicates that the “default” Index Service should be used for discovery queries. A second constructor, shown in line 5 of Figure 3-10, takes a String as an argument, and the String is expected to represent the service URL of the Index Service to query. The final constructor, not shown, takes an `EndPointReferenceType`, which can be used to directly indicate the Index Service Resource to query. The standard caGrid Index Service installation is stateless, and so a resource unqualified EPR can be used, but most clients can just use the shortcut String constructor.

```
1  public static void main(String[] args) {
2      DiscoveryClient client = null;
3      try {
4          if (args.length == 1) {
5              client = new DiscoveryClient(args[0]);
6          } else {
7              client = new DiscoveryClient();
8          }
9      } catch (MalformedURIException e) {
10         System.err.println("Invalid URI specified for Index Service:" + e);
11         e.printStackTrace();
12         System.exit(-1);
13     }
}
```

Figure 3-10 DiscoveryClient constructor example

The Index Service to use can also be reconfigured at runtime, by invoking the `setIndexEPR` method, shown in line 11 of Figure 3-11. Just as specifying the Index Service in the constructor generates an exception if the Address is not valid, so will the setter method.

```

1     EndpointReferenceType indexEPR = new EndpointReferenceType();
2     try {
3         Address address = new Address(
4             "http://cagrid01.bmi.ohio-state.edu:8080/wsrf/services/DefaultIndexService");
5         indexEPR.setAddress(address);
6     } catch (MalformedURLException e) {
7         System.err.println("Invalid URI specified for Index Service:" + e);
8         e.printStackTrace();
9         System.exit(-1);
10    }
11    client.setIndexEPR(indexEPR);

```

Figure 3-11 Configuring Index Service code

Discovering Services

Once a DiscoveryClient is configured, it can be continually used to discover services of interest. While the client is technically thread safe as long as the Index Service is not reconfigured during use, it is recommended a new DiscoveryClient instance is used in each thread context where discovery operations are performed, as it is an extremely light weight object.

The simplest discovery scenario, shown in Figure 3-12, is to query the Index Service for all registered services. The boolean value specified in line 3, indicates whether services should be ignored if they do not expose the caGrid standard metadata. In most application scenarios, a value of “true” is used, but specifying “false” is useful for identifying all services that are attempting to register. It is common for a service running behind a firewall to maintain registration status with the Index Service, but not have caGrid metadata aggregated, as the Index Service is not able to communicate with the inaccessible service

```

1     EndpointReferenceType[] allServices = null;
2     try {
3         allServices = client.getAllServices(true);
4     } catch (ResourcePropertyRetrievalException e1) {
5         e1.printStackTrace();
6         System.exit(-1);
7     }

```

Figure 3-12 Discovering all services

There are numerous discovery operations which take some form of text input, and all are case sensitive. The simplest discovery operation that takes some form of input is the basic string search operation, discoverServicesBySearchString, which is shown in Figure 3-13. This is a full text search that examines all registered metadata values for the specified input. It is not likely this operation will be useful for programmatic discovery (as it is a completely unstructured query), but it is useful for applications that take direct input from the user (such as a web form), and makes a good starting point for applications that provide capability to “drill down” and examine the full metadata of the satisfying services.

```

1     EndpointReferenceType[] services = null;
2     try {
3         services = client.discoverServicesBySearchString("chemical reaction");
4     } catch (ResourcePropertyRetrievalException e) {
5         e.printStackTrace();
6         System.exit(-1);
7     }

```

Figure 3-13 Discovering by Search String

Beyond the full text search operation, there are many discovery operations that take a search string as input, but perform a more structured search and are more useful for programmatic discovery. For example, services providing a named operation can be discovered using the method `discoverServicesByOperationName`, or Data Services exposing a given model can be discovered, as shown below in Figure 3-14, using the `discoverDataServicesByDomainModel` method. This operation, and all methods named like `discoverDataServices*` only return services that implement the standard Data Service operations.

```
1  EndpointReferenceType[] services = null;
2  try {
3      services = client.discoverDataServicesByDomainModel("caCORE");
4  } catch (ResourcePropertyRetrievalException e) {
5      e.printStackTrace();
6      System.exit(-1);
7 }
```

Figure 3-14 Discovering Data Services by Model Name

Another potentially useful method for discovering services or displaying information about available services on the grid is the `discoverServicesByResearchCenter` method, shown below in Figure 3-15.

```
1  EndpointReferenceType[] services = null;
2  try {
3      services = client.discoverServicesByResearchCenter("Ohio State University");
4  } catch (ResourcePropertyRetrievalException e) {
5      e.printStackTrace();
6      System.exit(-1);
7 }
```

Figure 3-15 Discovering by Research Center

There are several discovery methods that support semantic discovery by allowing search on concept code. The simplest of these methods, `discoverServicesByConceptCode`, shown below in Figure 3-16, searches for services based on concepts applied to the services itself. There is a concept representing “Grid Service” in the ontology and derived concepts such as “Analytical Grid Service” and “Data Grid Service.” By determining these concept codes, or any other specialized concepts, this operation provides a simple way to discover services of a certain “type.” Similarly, there is a method to discover services by the semantics of the operations they provide using the `discoverServicesByOperationConceptCode` method. At the time of this writing, services operations are not yet semantically annotated, but are expected to be soon. Finally, two methods: `discoverDataServicesByModelConceptCode` and `discoverServicesByDataConceptCode` provide the capability to discover services based on the information about the data types they operate over. Both examine the semantic information of the UML Classes used by the services. The first, `discoverDataServicesByModelConceptCode`, locates Data Services that are exposing access to data based on the concept. The second, `discoverServicesByDataConceptCode`, locates services that directly produce or consume data based on the concept. In both cases, the concept is considered a match if the Class is based on the concept or one of its attributes, attribute value domains, or enumerated value meanings. These methods are all based on direct concept matching; not only ontological operations. However,

these methods coupled with the EVS grid service, provide a powerful ability to traverse the caBIG ontology for information of interest, and discover services providing this information, or the ability to manipulate it.

```

1   EndpointReferenceType[] services = null;
2   try {
3       services = client.discoverServicesByConceptCode ("C43418");
4   } catch (ResourcePropertyRetrievalException e) {
5       e.printStackTrace();
6       System.exit(-1);
7   }

```

Figure 3-16 Discovering Services by Concept Code

Beyond the simple String based discovery methods, some discovery methods take complex objects as input, such as a `PointOfContact` or `UMLClass`. In these cases, the objects act as a prototype (or “query by example” as in the caCORE APIs), and can be as partially populated as desired. For example, the method shown below in Figure 3-17, `discoverServicesByPointOfContact`, searches for services which are associated with a person with the information described by the supplied `PointOfContact` instance; in this case services associated with “Scott Oster” are located. There are many other fields in `PointOfContact` that are not populated in this example, and are ignored.

```

1   EndpointReferenceType[] services = null;
2   try {
3       PointOfContact poc = new PointOfContact();
4       poc.setFirstName("Scott");
5       poc.setLastName("Oster");
6
7       services = client.discoverServicesByPointOfContact(poc);
8   } catch (ResourcePropertyRetrievalException e) {
9       e.printStackTrace();
10      System.exit(-1);
11  }

```

Figure 3-17 Discover Services by Point of Contact

There are many discovery methods that take a `UMLClass` prototype to discover services based on data types; an example is shown below in Figure 3-18. This method, `discoverServiceByOperationInput`, locates services that provide an operation that takes, as input, an instance of the specified data type. In the example below, services provide operations taking caBIO’s Gene instances as input. Again, this object can be as partially populated as desired (such as only specifying the package name, or being more explicit in specifying the exact project name and version).

```

1   EndpointReferenceType[] services = null;
2   try {
3       UMLClass umlClass = new UMLClass();
4       umlClass.setClassName("Gene");
5       umlClass.setPackageName("gov.nih.nci.cabio.domain");
6
7       services = client.discoverServicesByOperationInput(umlClass);
8   } catch (ResourcePropertyRetrievalException e) {
9       e.printStackTrace();
10      System.exit(-1);
11  }

```

Figure 3-18 Discover Services by Input

caDSR Grid Service Usage Overview

The following link provides a reference to the technical architecture and design document(s) for the caDSR Grid Service:

<http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/~checkout~/cagrid-1-0/Documentation/docs/cadsr/caGrid-cadsr-design.doc?rev=1.7;content-type=application%2Foctet-stream;cvsroot=cagrid-1-0>

The caGrid caDSR Grid Service provides access to information in the caDSR that is relevant to caGrid, and has capabilities to generate caGrid standard metadata instances. Specifically, the service provides operations to access UML-like information stored in the caDSR. It also has operations to generate Data Service metadata for a described subset of a given project registered in caDSR. Finally, it has an operation that augments a description of an Analytical Service, via a partially populated service metadata instance, with the necessary UML-like and semantic information, extracted from caDSR, to describe the service and its operations.

The caDSR Grid Service is a simple, stateless service, created with Introduce. The service exposes three main categories of operations. The first are operations that expose access to the UML-like view of caDSR registered items like `findProjects`, `findPackagesInProject`, `findClassesInPackage`, `findAttributesInClass`, etc. These provide basic discovery and access to the UML information in the caDSR. While these operations are stateless, they take sufficient context during each invocation to enable traversal of all registered projects. Aside from the operations to locate Projects, each operation takes a description of the caDSR Project of interest. Each operation in turn throws an `InvalidProjectException` if the Project specified is not valid.

The second set of operations enables clients to generate caGrid standard Data Service metadata. There are four operations that take variations of information specifying what data is to be exposed by the Data Service for which the metadata is being created. Each operation throws an `InvalidProjectException` if the Project specified is not valid or if it ambiguously identifies more than one Project (for example, if a version is not specified, yet there are multiple versions of a given Project registered in caDSR). The first operation, `generateDomainModelForProject`, takes only the project description and generates a model that describes the entire domain model being exposed for the project. The second, `generateDomainModelForPackage`, additionally takes an array of Strings that represent UML package names in the Project to expose. The method generates a model that describes exposing all UML Classes in UML Packages with a name specified in the array. Any associations to UML Classes outside of the specified packages are not exposed. The third method, `generateDomainModelForClasses`, also takes an array of Strings that represent the fully qualified UML Class names to be exposed in the model. Any association between classes not specified is omitted. The final method, `generateDomainModelForClassesWithExcludes`, also takes an additional array of Strings that represent the fully qualified UML Class names to be exposed in the model, but also takes an array of `UMLAssociationExcludes` to be used to exclude specific associations from the model (in addition to the already excluded associations that reference classes not specified in the array of class names). The `UMLAssociationExclude` Class allows the client to specify a `sourceRoleName`, `sourceClassName`, `targetRoleName`, and `targetClassName`. Any UML Association that would otherwise be included in the computed subset of the DomainModel is omitted if it meets the criteria described by any of the `UMLAssociationExcludes`. The value of any attribute of the `UMLAssociationExclude` can be the wildcard "*", which indicates it should match anything. As such, specifying an

exclude with “*” as the value for all attributes effectively omits all associations from the DomainModel. By using no wildcards, a single association can be omitted, and by using a combination of some values and some wildcards, groups of associations can be omitted. For example, specifying an exclude instance with a `sourceClassName` value of “`gov.nih.nci.cabio.domain.Gene`” and wildcards for all other attributes would effectively omit any associations from the DomainModel where `gov.nih.nci.cabio.domain.Gene` was the source of the association. Using these methods, in combination with the operations for finding all Projects, Packages, Classes, and Associations, Data Service metadata exposing any subset of Classes and Associations can be created.

The final type of operation is the operation, `annotateServiceMetadata`, which provides clients the ability to augment a `ServiceMetadata` (standard caGrid service metadata) skeleton instance with the information extracted from caDSR. The caGrid common service metadata specifies information about a grid service and its operations. For more information on the model, consult the caGrid metadata design document. The `annotateServiceMetadata` operation takes this model and populates the UML and semantically oriented components by querying the caDSR appropriately. Specifically, it populates the semantically annotated UML Class information (similar to the type used in Data Service Domain Model metadata) for each input and output type of every operation the service provides. It does this by examining the XML Qualified Name (QName) of each type used in the signature of the operation and locating its UML equivalent in caDSR. In caGrid every grid service operation is required to use data types which are XML representations of UML Classes registered in the caDSR. There is a one to one mapping of UML Class to XML QNames (XML elements). The caGrid Metadata Design Document and caDSR Grid Service Design Document can be consulted for more information on how this binding (XML QName to caDSR type) occurs, and what restrictions it places on the models.

The primary data types used by the caDSR grid service are those that are defined in caCORE 3.1 in the `gov.nih.nci.cadsr.domain` model, which represents the caDSR information and the `gov.nih.nci.cadsr.umlproject.domain`, which represents a UML-like view of information in the caDSR. The `umlproject` model, shown below in Figure 3-19, is the main model but it associates with and extends a few classes from the caDSR base model, so it is used as well. As is evident from the figure below, the model provides a UML-like view of the caDSR registered projects. One class of note is the `SemanticMetadata` class which is associated to many UML-like classes, and provides a link to the semantic content of those items. Specifically, it exposes information about the EVS-maintained concepts.

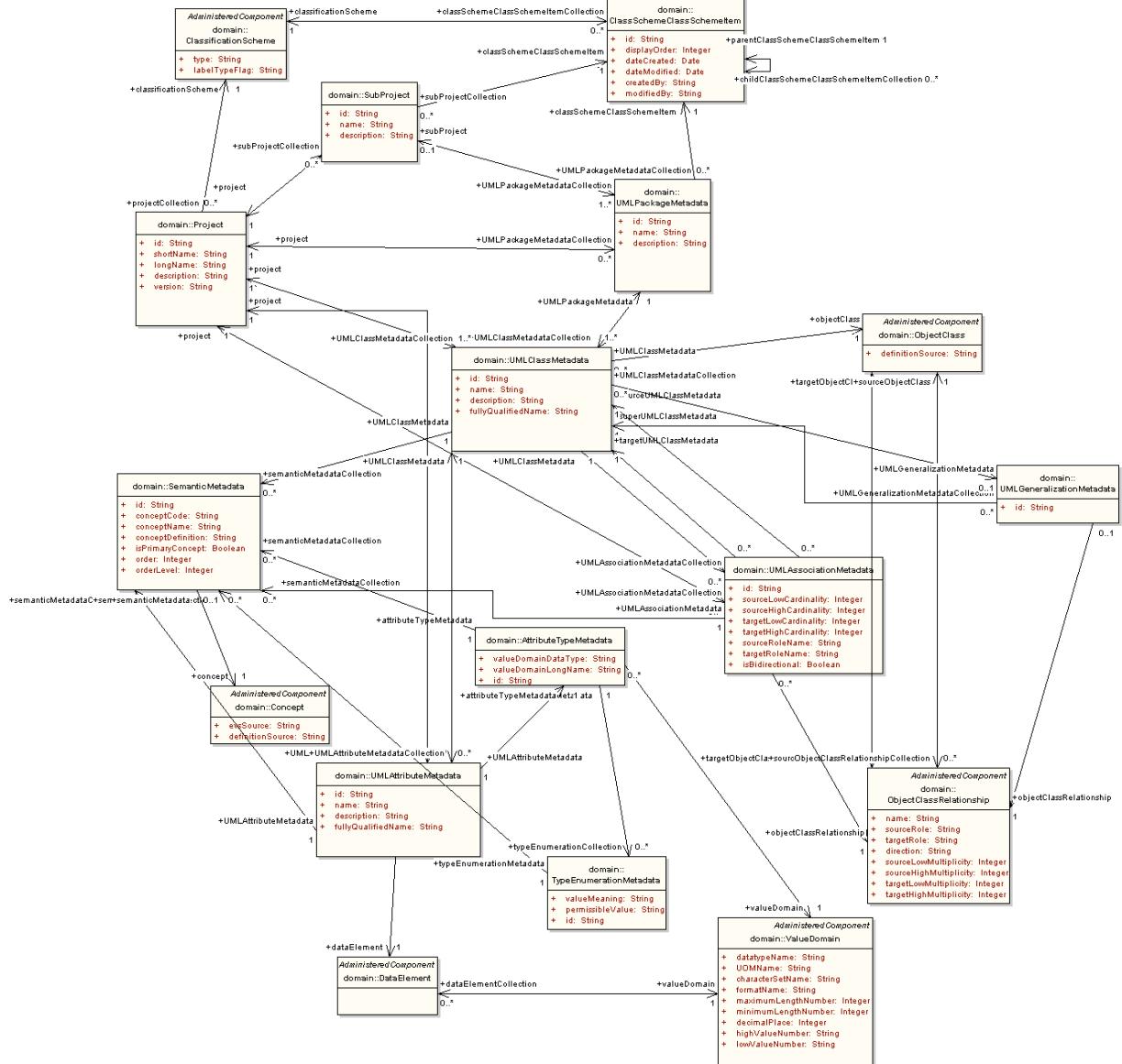


Figure 3-19 UML project model

In addition to the existing caCORE-defined types, the caDSR grid service defines two new data types for exclusive use by its exposed operations. The first is the `UMLAssociationExclude`, described above as the type used to specify UML Associations which should be excluded from a generated `DomainModel`. The second is an alternative representation of a UML Association, namely the `UMLAssociation` class. This has the same semantics as the `UMLAssociationMetadata` class in the `umlproject` model, but uses an alternate syntactic representation which is more suitable to transport over the grid.

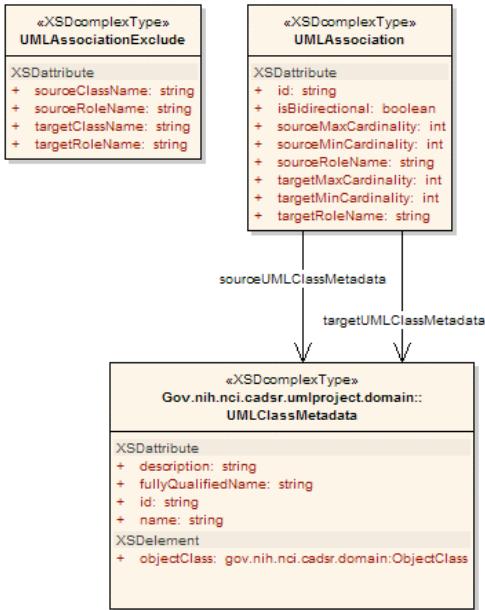


Figure 3-20 caDSR Grid Service Types

Finally the service also makes use of the `ServiceMetadata` and `DomainModel` caGrid metadata models, as it provides operations to manipulate them. For more information on these models, consult the [caGrid Metadata Design Document](#).

Security Considerations

The caDSR grid service requires no grid credentials for any operations. Its typical deployment is in a service container using an open communication channel. However, even if it is deployed to a container making use of transport level security (https), it will not require credentials from the user and can be communicated with anonymously.

API Details

The `caDSRServiceClient` is the main client interface to the caDSR grid service (Figure 3-21).

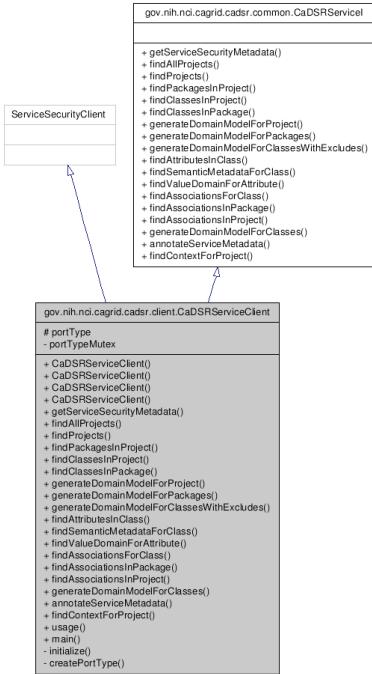


Figure 3-21 *CaDSRServiceClient Inheritance Graph*

`gov.nih.nci.cagrid.cadsr.client.CaDSRServiceClient`

Constructor Documentation

CaDSRServiceClient (String url) - throws MalformedURLException, RemoteException

CaDSRServiceClient (String url, GlobusCredential proxy) - throws MalformedURLException, RemoteException

CaDSRServiceClient (EndpointReferenceType epr) - throws MalformedURLException, RemoteException

CaDSRServiceClient (EndpointReferenceType epr, GlobusCredential proxy) - throws MalformedURLException, RemoteException

Member Function Documentation

Access to caGrid Service Security Metadata:

- `gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata getServiceSecurityMetadata ()`
 - throws RemoteException
 - **Description:** Returns the caGrid service security metadata

Navigation and discovery of UML-like information:

- `gov.nih.nci.cadsr.umlproject.domain.Project[] findAllProjects ()`
 - throws RemoteException
 - **Description:** Returns all Projects registered in the caDSR

- gov.nih.nci.cadsr.umlproject.domain.Project[] **findProjects** (java.lang.String context)
 - throws RemoteException
 - **Description:** Returns all Projects registered in the caDSR under the given context
- gov.nih.nci.cadsr.umlproject.domain.UMLPackageMetadata[] **findPackagesInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns all packages in the given Project
- gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata[] **findClassesInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns all Classes in the given Project
- gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata[] **findClassesInPackage** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String packageName)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns all Classes in the given Package
- gov.nih.nci.cadsr.umlproject.domain.UMLAttributeMetadata[] **findAttributesInClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns all Attributes of the given Class
- gov.nih.nci.cadsr.umlproject.domain.SemanticMetadata[] **findSemanticMetadataForClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns the semantic information about the given Class
- gov.nih.nci.cadsr.domain.ValueDomain **findValueDomainForAttribute** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLAttributeMetadata attribute)
 - throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
 - **Description:** Returns the Value Domain information for the given Attribute

- gov.nih.nci.cagrid.cadrservice.UMLAssociation[] **findAssociationsForClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Returns all Associations of the given Class
- gov.nih.nci.cagrid.cadrservice.UMLAssociation[] **findAssociationsInPackage** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String packageName)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Returns all associations in the given package
- gov.nih.nci.cagrid.cadrservice.UMLAssociation[] **findAssociationsInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Returns all Associations in the given Project
- gov.nih.nci.cadsr.domain.Context **findContextForProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Returns the Context of the given Project

caGrid standard metadata generation and manipulation:

- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Generates standard Data Service metadata for the given Project
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForPackages** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] packageNames)
 - throws RemoteException, gov.nih.nci.cagrid.cadrsr.stubs.types.InvalidProjectException
 - **Description:** Generates standard Data Service metadata for the given Packages
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForClasses** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] fullClassNames)

- throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
- **Description:** Generates standard Data Service metadata for the given Classes
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel
generateDomainModelForClassesWithExcludes
 (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] fullClassNames, gov.nih.nci.cagrid.cadsrservice.UMLAssociationExclude[] associationExcludes)
- throws RemoteException,
gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
- **Description:** Generates standard Data Service metadata for the given Classes, excluding the specified (if any) Associations
- gov.nih.nci.cagrid.metadata.ServiceMetadata **annotateServiceMetadata**
 (gov.nih.nci.cagrid.metadata.ServiceMetadata serviceMetadata)
- throws RemoteException
- **Description:** Annotates the given standard service metadata instance with semantically annotated UML class information for all operations inputs and outputs

API Usage Examples

This section describes typical usage of the caDSR Grid Service Client API. The exception handling shown in the code examples is not recommended practice, and is simplistic for demonstration purposes.

Examining a Project's Information Model

The example shown below in Figure 3-22, shows simple logic that can be used to list out the Projects, Packages, Classes, and Attributes registered in the caDSR. It is a simplified extract of the main method of the CaDSRServiceClient, and can run in full from the project by typing “ant runClient.” The first step in communicating with the service, as with all caGrid services, is to construct an instance of the client, passing the URL of the service. This can be seen in line 1 of the example. Next in line 4, the example asks the caDSR for all of the registered Projects, and then loops over them in line 6. For each returned Project, the caDSR is then asked for the Packages registered in that Project, shown in line 9. This demonstrates the basic process which is used to navigate the UML-like information in the caDSR, wherein context of the previous operation (in this case a Project), is passed to a successive operation which provides more detailed information. This can be seen again in line 15, line 22, and line 31, where respectively the Classes in a Package, Attributes in a Class, and Associations in a Class are accessed. The result of running this example is a printout out of a “tree like” view of all caDSR registered Projects. While generally one does not want to navigate all information in the caDSR, this basic model can be followed to find whatever information is desired (first locating the Project of interest, then passing it to operations which provide more detail).

```
1  CaDSRServiceClient cadsr = new CaDSRServiceClient(
2      "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4  Project[] projs = client.findAllProjects();
5  if (projs != null) {
6      for (int i = 0; i < projs.length; i++) {
7          Project project = projs[i];
8          System.out.println("\n" + project.getShortName());
9          UMLPackageMetadata[] packs = client.findPackagesInProject(project);
10         if (packs != null) {
11             for (int j = 0; j < packs.length; j++) {
12                 UMLPackageMetadata pack = packs[j];
13                 System.out.println("\t" + pack.getName());
14
15                 UMLClassMetadata[] classes = client.findClassesInPackage(
16                     project, pack.getName());
17                 if (classes != null) {
18                     for (int k = 0; k < classes.length; k++) {
19                         UMLClassMetadata clazz = classes[k];
20                         System.out.println("\t\t" + clazz.getName());
21                         UMLAttributeMetadata[] atts =
22                             client.findAttributesInClass(project, clazz);
23                         if (atts != null) {
24                             for (int l = 0; l < atts.length; l++) {
25                                 UMLAttributeMetadata att = atts[l];
26                                 System.out.println("\t\t\t" + att.getName());
27                             }
28                         }
29
30                         UMLAssociation[] assocs =
31                             client.findAssociationsForClass(project, clazz);
32                         if (assocs != null) {
33                             for (int index = 0; index < assocs.length; index++) {
34                                 UMLAssociation assoc = assocs[index];
35                                 System.out.println("\t\t\t"
36                                     + assoc.getSourceRoleName()
37                                     + "----> (" + assoc.getTargetRoleName() + ")"
38                                     + assoc.getTargetUMLClassMetadata()
39                                         .getUMLClassMetadata().getFullyQualifiedName());
40                             }
41                         }
42                     }
43                 }
44             }
45         }
46     }
47 }
48 }
```

Figure 3-22 Navigating projects registered in the caDSR

Generating Data Service Metadata

As described above, there are four operations which provide the ability to generate instances of caGrid standard Data Service metadata. Three examples of using this capability are shown below, wherein the resulting DomainModel is written to a local file, and some basic information about the model is printed to the screen. It should be noted that most users will not need to make use of these APIs, as the Introduce toolkit automatically does this for them when creating a Data Service, using the appropriate information based on the configuration of the service.

The first example, shown in Figure 3-23, demonstrates how to create metadata which describes an entire Project being exposed; in this case the 3.1 version of caCORE.

Lines 4-6 demonstrate the construction of a “prototype” Project instance, which describes the Project being exposed. Then, in line 8, this is passed to the `generateDomainModelForProject` operation of the caDSR grid service, and the resulting metadata instance is returned. This is the simplest of the Data Service metadata generation operations, and does not allow for configuration or restriction of the model. However, the resulting object could then be manipulated by the client appropriately.

The following lines (10-19) are common to all the following examples and show how the DomainModel can be written to a local file (lines 10-13), and how the model can be inspected as a Java Bean. Line 17 shows how the descriptions of the Associations of the model are accessed, and line 19 shows how the descriptions of the exposed Classes can be accessed.

```

1  CaDSRServiceClient cadsr = new CaDSRServiceClient(
2      "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4  Project project = new Project();
5  project.setVersion("3.1");
6  project.setShortName("caCORE");
7
8  DomainModel domainModel = cadsr.generateDomainModelForProject(project);
9
10 FileWriter fw = new FileWriter(project.getShortName()
11     + "_" + project.getVersion() + "_DomainModel.xml");
12 MetadataUtils.serializeDomainModel(domainModel, fw);
13 fw.close();
14
15 System.out.println("Domain Model generation complete.");
16 System.out.println("\t exposed Association count:"
17     + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
18 System.out.println("\t exposed Class count:"
19     + domainModel.getExposedUMLClassCollection().getUMLClass().length);

```

Figure 3-23 Generating a Domain Model for a Whole Project

The next example, shown in Figure 3-24, demonstrates how a metadata instance that describes exposing a subset of a Project can be created. In this case, on line 9, an array of package names, containing only caBIO, is constructed and passes to the `generateDomainModelForPackages` operation. This operation restricts the returned model to only contain Classes that are in the specified package(s), and Associations between those Classes.

```
1  CaDSRServiceClient cadsr = new CaDSRServiceClient(
2      "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4  Project project = new Project();
5  project.setVersion("3.1");
6  project.setShortName("caCORE");
7
8  DomainModel domainModel =cadsr.generateDomainModelForPackages(project,
9      new String[]{"gov.nih.nci.cabio.domain"});
10
11 FileWriter fw = new FileWriter(project.getShortName()
12      + "_" + project.getVersion() + "_DomainModel.xml");
13 MetadataUtils.serializeDomainModel(domainModel, fw);
14 fw.close();
15
16 System.out.println("Domain Model generation complete.");
17 System.out.println("\t exposed Association count:"
18      + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
19 System.out.println("\t exposed Class count:"
20      + domainModel.getExposedUMLClassCollection().getUMLClass().length);
```

Figure 3-24 Generating a domain model for a package

The final example, shown in Figure 3-25, demonstrates the most powerful domain model generation operation, which allows the explicit specification of the Classes which are being exposed and the restriction of Associations between those Classes. Shown in lines 8 and 9, an array of Class names is constructed, in this case the generated model will specify the Data Service is only exposing 4 classes: Gene, Chromosome, Taxon, and Tissue. If we generated a model using these Classes, all Associations between them would be included. However, in the example some of these Associations are specified to be excluded from the model, signifying clients may not query over these associations. Lines 10 and 11, first show the construction of an `AssociationExclude` that explicitly specifies to exclude the Association from Gene to Chromosome where Gene is the source, with the role “geneCollection” and Chromosome is the target with “chromosome” as the role. This only matches a single association in the model. Next, in lines 12-13, a broader exclude is constructed which specifies that an Association with Tissue as the target should be excluded. Both these restrictions are then added to an array, in lines 14 and 15, and passed to the service with the Project and Class names on line 16.

```

1  CaDSRServiceClient cadsr = new CaDSRServiceClient(
2      "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4  Project project = new Project();
5  project.setVersion("3.1");
6  project.setShortName("caCORE");
7
8  String classNames[] = new String[]{Gene.class.getName(), Chromosome.class.getName(),
9      Taxon.class.getName(), Tissue.class.getName()};
10 UMLAssociationExclude exclude1 = new UMLAssociationExclude(Gene.class.getName(),
11      "geneCollection", Chromosome.class.getName(), "chromosome");
12 UMLAssociationExclude exclude2 = new UMLAssociationExclude("*", "*",
13      Tissue.class.getName(), "*");
14 UMLAssociationExclude associationExcludes[] = new UMLAssociationExclude[]{
15      exclude1, exclude2};
16 DomainModel domainModel = cadsr.generateDomainModelForClassesWithExcludes(project,
17      classNames, associationExcludes);
18
19  FileWriter fw = new FileWriter(project.getShortName()
20      + "_" + project.getVersion() + "_DomainModel.xml");
21  MetadataUtils.serializeDomainModel(domainModel, fw);
22  fw.close();
23
24  System.out.println("Domain Model generation complete.");
25  System.out.println("\t exposed Association count:"
26      + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
27  System.out.println("\t exposed Class count:"
28      + domainModel.getExposedUMLClassCollection().getUMLClass().length);

```

Figure 3-25 Generating a restricted domain model

EVS API Usage Overview

The following link provides a reference to the technical architecture and design document(s) for the EVS API:

<http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/~checkout~/cagrid-1-0/Documentation/docs/evs/EVS%20Grid%20Service%20Design%20and%20Implementation.doc?rev=1.4;content-type=application%2Foctet-stream;cvsroot=cagrid-1-0>

getMetaSources

This API provides a list of vocabularies presented by the NCI Metathesaurus. The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing and data pooling for clinical trials and scientific data services. The Metathesaurus is based on the National Library of Medicine's (NLM) Unified Medical Language System (UMLS) and is composed of over 70 biomedical vocabularies.

Input:

None

Output:

gov.nih.nci.evs.domain.Source[]

Exception:

RemoteException

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);  
gov.nih.nci.evs.domain.Source[] sources = client.getMetaSources();
```

where `endpoint` provides an **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid.

getVocabularyNames

This API provides a list of vocabularies whose concepts are programmatically accessible to the users via the Description Logic representation. All the vocabularies that are accessible via the caCORE 3.1 EVS API are supported by this API.

Input:

None

Output:

```
gov.nih.nci.cagrid.evs.service.DescLogicConceptVocabularyName []
```

Exception:

RemoteException

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);  
gov.nih.nci.cagrid.evs.service.DescLogicConceptVocabularyName[] dlcNames =  
client.getVocabularyNames();
```

where `endpoint` provides an **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid.

searchDescLogicConcept

This API provides access to concepts and terms that are published by the vocabularies using the Description Logic representation and exposed via the caCORE 3.1 EVS API.

The input to the API (`EVSDescLogicConceptSearchParams`) consists of appropriate vocabulary to query for concepts (`vocabularyName`), the concept name or code to search (`searchTerm`) and a tuning parameter (`limit`) to restrict the amount of objects returned by the API.

The API returns an array of Description Logic concepts with most of the attributes populated. The user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

```
gov.nih.nci.cagrid.evs.service.EVSDescLogicConceptSearchParams
```

Output:

```
gov.nih.nci.evs.domain.DescLogicConcept []
```

Exception:

```
 RemoteException
 InvalidInputExceptionType
```

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSDescLogicConceptSearchParams evsSearchParams = new
EVSDescLogicConceptSearchParams();
    evsSearchParams.setVocabularyName("NCI_Thesaurus");
    evsSearchParams.setSearchTerm(searchTerms[count]);
    evsSearchParams.setLimit(100);
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);
DescLogicConcept[] dlc= client.searchDescLogicConcept(evsSearchParams);
```

where `endpoint` provides an **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid.

API-Specific Considerations

The API has the following restrictions:

1. Instances of `gov.nih.nci.evs.domain.EdgeProperties` are not populated currently by the API. These objects are used to specify a relationship between a concept and its immediate parent when a `DefaultMutableTree` is generated. The caGrid EVS 1.0 service is not required to support the generation of a Tree.
2. Instances of `gov.nih.nci.evs.domain.Qualifier` are not populated by the caGrid EVS 1.0 service.
3. Instances of `gov.nih.nci.evs.domainTreeNode` are not populated by the caGrid EVS 1.0 service.

Exception Handling

The API validates inputs and throws `InvalidInputExceptionType` if any of the following invalid values are present in the input to the API:

1. If the input instance `EVSDescLogicConceptSearchParams` is null.
2. If the value of the tuning parameter(`limit`) is less than or equal to zero.
3. If vocabulary name (`vocabularyName`) is not set (null or empty string) or the vocabulary name is not present in the list of available vocabularies supported by caCORE 3.1 EVS API for concepts based on Description Logic representation.

getHistoryRecords

This API provides a complete History for concepts and traces the evolution of concepts as they are created, merged, modified, split, or retired. This history mechanism is provided completely for the NCI Thesaurus and is published by the NCI EVS team. For all other vocabularies that provide concepts based on Description Logic representation, a dummy value for History is provided.

The input to the API (`EVSHistoryRecordsSearchParams`) consists of appropriate vocabulary to query for concepts (`vocabularyName`) and a valid concept code to search (`conceptCode`).

The API returns an array of History records for the specified Description Logic Concept with attributes populated. The user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

```
gov.nih.nci.cagrid.evs.service.EVSHistoryRecordsSearchParams
```

Output:

```
gov.nih.nci.evs.domain.HistoryRecord[]
```

Exception:

```
RemoteException
```

```
InvalidInputExceptionType
```

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSHistoryRecordsSearchParams evsHistoryParams = new  
EVSHistoryRecordsSearchParams();  
evsHistoryParams.setVocabularyName("NCI_Thesaurus");  
evsHistoryParams.setConceptCode("C16612");  
  
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);  
HistoryRecord[] historys = client.getHistoryRecords(evsHistoryParams);
```

where `endpoint` provides an **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid.

API-Specific Considerations

None

Exception Handling

The API validates inputs and throws `InvalidInputExceptionType` if any of the following invalid values are present in the input to the API:

- If the input instance `EVSHistoryRecordsSearchParams` is null.
- If the concept code (`conceptCode`) is not set (null or empty string) or an invalid concept code is passed to the API. All concept codes are expected to start with the letter "C".
- If the vocabulary name (`vocabularyName`) is not set (null or empty string) or the vocabulary name is not present in the list of available vocabularies supported by caCORE 3.1 EVS API for concepts based on Description Logic representation.

searchMetaThesaurus

This API provides access to concepts that are supported by the NCI Metathesaurus. The input to the API (`EVSMetaThesaurusSearchParams`) consists of a search term or a concept unique identifier (CUI) (`searchTerm`), a valid Metathesaurus source (`source`), and tuning parameters to control the amount of results (`limit`, `shortResponse` and `score`). The API returns an array of concepts from the Metathesaurus with all the attributes populated

and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

```
gov.nih.nci.cagrid.evs.service.EVSMetaThesaurusSearchParams
```

Output:

```
gov.nih.nci.evs.domain.MetaThesaurusConcept[]
```

Exception:

```
RemoteException  
InvalidInputExceptionType
```

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSMetaThesaurusSearchParams evsMetaThesaurusSearchParam = new
EVSMetaThesaurusSearchParams();
evsMetaThesaurusSearchParam.setLimit(100);
evsMetaThesaurusSearchParam.setSource("*");
evsMetaThesaurusSearchParam.setCui(false);
evsMetaThesaurusSearchParam.setShortResponse(false);
evsMetaThesaurusSearchParam.setScore(false);
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);
MetaThesaurusConcept[] metaConcept =
client.searchMetaThesaurus(metaParams);
```

where `endpoint` provides an **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid.

API-Specific Considerations

The API has the following constraints:

- The source input to the API has to be a valid metathesaurus source abbreviation (present in the list from the API `getMetaSources`. A value of “*” indicates that the search term will be queried against ALL available sources.
- Search term refers to the concept “name” to be searched in the appropriate source or all sources as indicated above.
- The appropriate settings of the tuning parameters to indicate whether the search term is a Concept Unique Identifier (CUI) or a regular search term. If the search term is a CUI, then, the search term is validated to adhere to the following restrictions:
 - The CUI begins with the letter “C”
 - The CUI has a maximum length of 8 characters

Exception Handling

The API validates inputs and throws `InvalidInputExceptionType` if any of the following invalid values are present in the input to the API:

1. If the input instance `EVSMetaThesaurusSearchParams` is null.
2. If the search term (`searchTerm`) is not set (null or empty).

3. If the search term is a CUI, then if an invalid CUI is passed to the API. All CUIs are expected to start with the letter “C” and are 8 characters long.
4. If the value of the tuning parameter (`limit`) is less than or equal to zero.
5. If the Metathesaurus source abbreviation (`source`) is not set (null or empty string) or the source is not present in the list of available vocabularies supported by caCORE 3.1 EVS API (“*” is valid source abbreviation indicating ALL sources)

searchSourceByCode

This API provides access to concepts that are supported by the NCI Metathesaurus. The input to the API (`EVSSourceSearchParams`) consists of a concept unique identifier (CUI) (`code`) and a valid Metathesaurus source (`source`). The API returns an array of concepts from the Metathesaurus with all the attributes populated and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

```
gov.nih.nci.cagrid.evs.service.EVSSourceSearchParams
```

Output:

```
gov.nih.nci.evs.domain.MetaThesaurusConcept[]
```

Exception:

```
RemoteException
```

```
InvalidInputExceptionType
```

Examples of Use

The following is an example of the java client code invoking the API:

```
EVSSourceSearchParams evsSourceParam = new EVSSourceSearchParams();  
evsSourceParam.setCode("0000001800");  
evsSourceParam.setSourceAbbreviation("AOD2000");  
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);  
MetaThesaurusConcept[] mc = client.searchSourceByCode(evsSourceParam);
```

where `endpoint` provides an **Endpoint Reference (EPR)** to the EVS core grid service deployed on caGrid.

API-Specific Considerations

The API will take following input:

- A valid MetaThesaurus source abbreviation (present in the list from the API `getMetaSources`). The ALL sources value of “*” is not permitted for this API.
- A valid value for “code”. Some of the concepts in the MetaThesaurus do not have code associated with them and are displayed in the NCI Meta Thesaurus Browser with the value of “NOCODE”. The “NOCODE” value is not valid input for the API.

Exception Handling

The API validates inputs and throws `InvalidInputExceptionType` if any of the following invalid values are present in the input to the API:

- If the input instance `EVSSourceSearchParams` is null.
- If the concept code (`code`) is not set (null or empty) or has the value “NOCODE”.
- If the Metathesaurus source abbreviation (`source`) is not set (null or empty string) or the source is not present in the list of available vocabularies supported by caCORE 3.1 EVS API.

Chapter 4 caGrid Security

This chapter describes using Dorian, Grid Grouper, and WebSSO as part of caGrid security.

Topics in this chapter include:

- Dorian Overview on this page
- Grid Grouper Overview on page 90
- WebSSO Server Installation on page 103

Dorian Overview

Managing users and provisioning accounts in the grid is complex. The Globus Toolkit implements support for security via its Grid Security Infrastructure (GSI). The GSI utilizes X509 Identity Certificates for identifying a user. An X509 Certificate with its corresponding private key forms a unique credential or so-called “grid credential” within the Grid. Since Grid credentials are long term credentials and are not directly used in authenticating users to the grid, a short term credential called a grid proxy is used instead. Grid proxies consist of a private key and a corresponding long term certificate signed by the long term grid credential private key. A grid proxy is an extension to traditional X509 certificates providing the ability to delegate your credentials to other services, as in the case of a workflow for example. Although this approach is effective and secure, it is difficult to manage in a multi-institutional environment., The provisioning of grid credentials is a manual process using the base Globus Toolkit, which is far too complicated for users. The overall process is further complicated if a user wishes to authenticate from multiple locations, as a copy of their private key and certificate has to be present at every location. Not only is this process complicated, securely distributing private keys is error prone and poses a security risk. There are also many complexities in terms of provisioning user accounts in an environment consisting of tens of thousands of users from hundreds of institutions, each of which most likely has a user account at their home institution. A practical solution to this problem for both users and their institutions is to allow users to authenticate with the grid through the same mechanism in which they authenticate with their institution. Dorian is a grid user management service that (1) hides the complexities of creating and managing grid credentials from the users and (2) provides a mechanism for users to authenticate using their institution’s authentication mechanism, assuming a trust agreement is in place between Dorian and the institution.

Dorian provides a complete Grid-enabled solution, based on public key certificates and SAML, for managing and federating user identities in a Grid environment. Grid technologies have adopted the use of X509 identity certificates to support user authentication. The Security Assertion Markup Language (SAML) has been developed as a standard for exchanging authentication and authorization statements between security domains. Note that Grid certificates and SAML assertions serve different purposes. SAML is mainly used between institutions for securely exchanging authentication information coming from trusted identity providers. The primary use of the certificates is to uniquely identify Grid users, facilitate authentication and authorization across multiple resource providers, and enable secure delegation of credentials such that a service or a client program can access resources on behalf of the user. A salient feature of Dorian is that it provides a mechanism for the combined use of both SAML

and Grid certificates to authenticate users to the Grid environment through their institution's authentication mechanism.

One of the challenges in building an identity management and federation infrastructure is to create an architecture that incorporates multiple differing authentication mechanisms used by various institutions. In addressing this challenge we identify two possible approaches. The first is to build an infrastructure that would allow pluggable authentication modules, wherein a module would be developed for each authentication mechanism. In this architecture, a user's authentication information would be routed to the appropriate module that contains the logic for authenticating the user with its institution. Although this approach solves the problem, it requires at least one module be developed for each authentication mechanism. This would require the Grid infrastructure administrators to become intimately familiar with each institution's authentication mechanisms, and would increase the system's complexity with each new module added.

Another approach is for the infrastructure to accept an institutionally-supplied, standard "token" as a method of authentication. In this approach users would first authenticate with their institution's identity management system. Upon successful authentication, the institution's identity management system issues a token which can then be given to the federated grid identity management system in exchange for grid credentials. The benefit of this approach over the first is that it does not require writing a plug-in every time a new institutional authentication mechanism comes online. It does, however, require every institutional authentication system to agree upon and be able to provide a common token. As SAML has been adopted by many institutions, we have chosen that token format as the basis of the second approach for Dorian.

The Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains. Generally the exchange of authentication and authorization data is made between an *Identity Provider* (IdP) and another party. An institution's authentication system or identity management system is an example of an IdP. Dorian uses SAML authentication assertions as the enabling mechanism for federating users from local institutions to the grid.

Figure 4-1 illustrates an example usage scenario for Dorian. To obtain grid credentials or a proxy certificate, users authenticate with their institution using the institution's conventional mechanism. Upon successfully authenticating the user, the local institution issues a digitally signed SAML assertion, vouching that the user has authenticated. The user then sends this SAML assertion to Dorian in exchange for grid credentials. Dorian will only issue grid credentials to users that supply a SAML assertion from a Trusted Identity Provider. Dorian's grid service interface provides mechanisms for managing trusted identity providers; this will be discussed in greater detail later in this document. For example, in Figure 4-1, when a Georgetown user wishes to invoke a grid service that requires grid credentials, they first supply the application with their username and password to the Georgetown Authentication Service as they would normally do. The application client authenticates the Georgetown user with the Georgetown Authentication Service, receives a signed SAML assertion which it subsequently passes to Dorian in exchange for grid credentials. These credentials can then be used to invoke the grid services. This illustrates how Dorian can leverage an institution's existing authentication mechanism and bring its users to the grid.

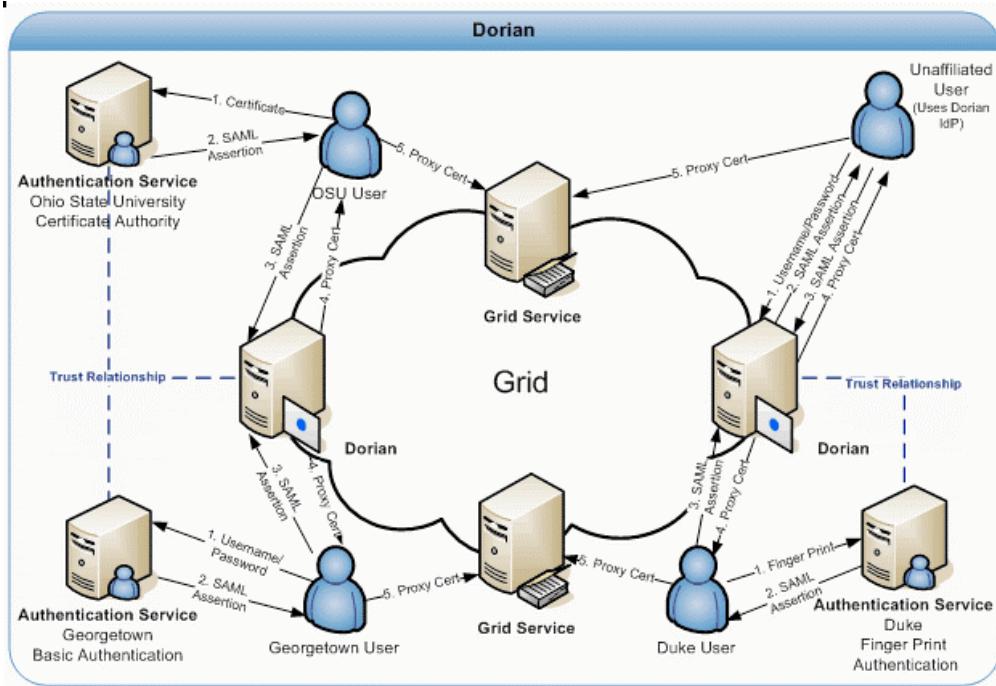


Figure 4-1 Dorian

To facilitate smaller groups or institutions without an existing IdP, Dorian also has its own internal IdP. This allows users to authenticate to Dorian directly, thereby enabling them to access the grid. It provides administrators with facilities for approving and managing users. All of Dorian IdP's functionality is made available through a grid service interface. Details of the Dorian IdP are provided later in this document. Figure 4-1 illustrates a scenario of a client using the Dorian IdP to authenticate to the Grid. In this scenario, the unaffiliated User wishes to invoke a grid service. Given that this unaffiliated user has registered and been approved for an account, she is able to authenticate with the Dorian IdP by supplying their username and password. Upon successfully authenticating the user, the Dorian IdP issues a SAML Assertion just like institutional IdPs, which can be presented to Dorian in exchange for grid credentials. The credentials can be used to invoke the grid service.

Creating a Grid Proxy Programmatically

Figure 4-2 provides an example of how to create a grid proxy programmatically with Dorian. In order to create a grid proxy using Dorian you must first obtain a signed SAML Assertion from an Identity Provider Trusted by Dorian. caGrid's Authentication Service provides a common interface and client tooling for exposing a local Identity Provider, such that a user may authenticate using their local credentials and obtain a SAML assertion using a common client or AuthenticationClient. Although it is not required to obtain the SAML Assertion from a caGrid Authentication Service, it is the recommended approach and the approach used in Figure 4-2. Besides obtaining a SAML assertion, Dorian also requires the specification of a proxy lifetime and a delegation path length in order to create a grid proxy. The proxy lifetime specifies the amount of time for which the proxy is valid. A proxy lifetime is specified in terms of hours, minutes, and seconds. The delegation path length specifies how many times a proxy can be delegated to other services. Once you have obtained a SAML Assertion and specified a proxy lifetime and

delegation path length, you can use Dorian's IFSUserClient to create a proxy with Dorian.

```
try{
    String authURI = "http://some.service.uri";
    String dorianURI = "http://some.dorian.uri";

    //Create an instance of my institution provided credentials
    Credential localCredential = new Credential();
    BasicAuthenticationCredential userPass = new
    BasicAuthenticationCredential();
    userPass.setUserId("MyUserId");
    userPass.setPassword("MyPassword");
    localCredential.setBasicAuthenticationCredential(userPass);

    //User the caGrid common authentication client to authenticate
    with the local
    //IdP and obtain a SAML Assertion

    AuthenticationClient auth = new AuthenticationClient(authURI,
    localCredential);
    SAMLAAssertion saml = auth.authenticate();

    //Specify the lifetime of the desired proxy
    ProxyLifetime lifetime = new ProxyLifetime();
    lifetime.setHours(12);
    lifetime.setMinutes(0);
    lifetime.setSeconds(0);

    //Specify the delegation path length of the desired proxy
    int delegationPathLength = 0;

    //Now Create a proxy using the Dorian IFS User Client
    IFSUserClient client = new IFSUserClient(dorianURI);
    GlobusCredential proxy = client.createProxy(saml, lifetime,
    delegationPathLength);

    //Now specify to Globus to use the aquired proxy as the default credentials
    ProxyUtil.saveProxyAsDefault(proxy);
} catch(Exception e){
    e.printStackTrace();
}
```

Figure 4-2 Programmatically creating a grid proxy with Dorian

Grid Grouper Overview

Grid Grouper provides a group-based authorization solution for caGrid, where grid services and applications enforce authorization policy based on membership to groups defined and managed at the grid level. Grid Grouper is built on top of Grouper, an internet2 initiative focused on providing tools for group management. Grouper is a java object model which currently supports: basic group management by distributed authorities; subgroups; composite groups (whose membership is determined by the union, intersection, or relative complement of two other groups); custom group types and custom attributes; trace back of indirect membership; and delegation. Applications interact with Grouper by embedding the Grouper's java object model within applications. Grouper does not provide a service interface for accessing groups. Grid Grouper (Figure 4-3) is a grid-enabled version of Grouper, providing a web service interface to the Grouper object model. Grid Grouper make groups managed by Grouper available and

manageable to applications and other services in the grid. Grid Grouper provides an almost identical object model to the Grouper object model on the grid client side. Applications and services can use the Grid Grouper object model much like they would use the Grouper object model to access and manage groups as well as enforce authorization policy based on membership to groups. Grid Grouper provides a fully functional administrative UI for accessing and administrating groups in Grid Grouper.

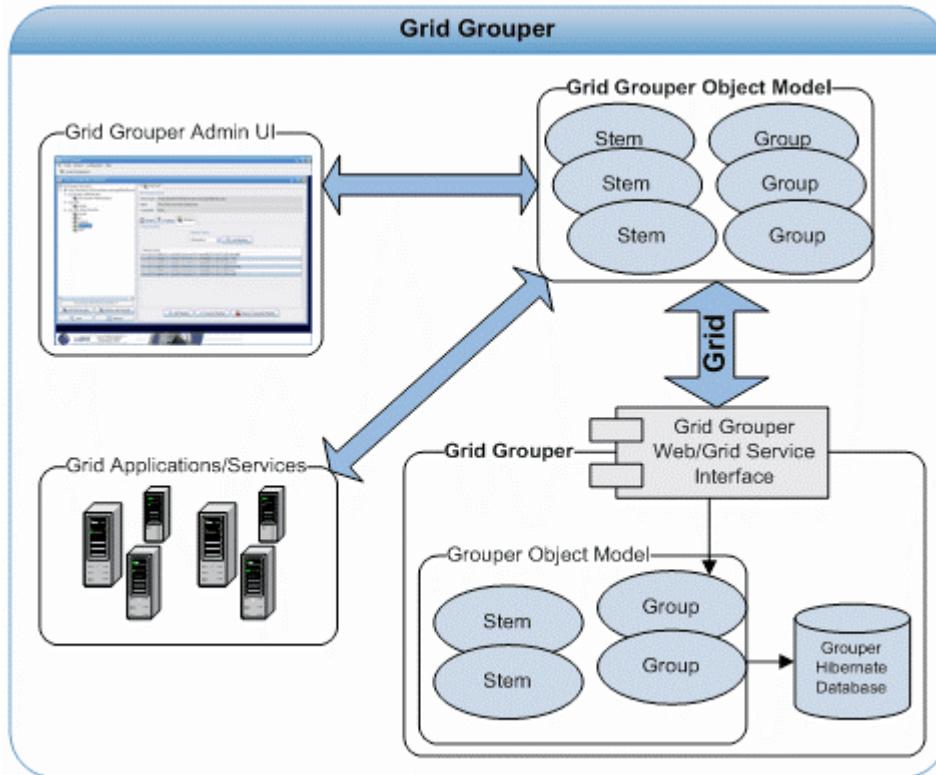


Figure 4-3 Grid Grouper

In Grouper/Grid Grouper groups are organized into namespaces or stems. Each stem can have a set of child stems and a set of child groups with exception to the root stem which cannot have any child groups. The Stem hierarchy in Grid Grouper is publicly visible to anyone accessing the service; however the ability to view a group within a stem publicly depends on the privileges for the group. A Stem can have two types of privileges associated with it; the “Stem Privilege” and the “Create Privilege”. Users with the “Stem Privilege” can create, modify, and remove child stems. Users with the “Create Privilege” can create, modify, and remove child groups.

In Grouper/Grid Grouper groups are comprised of a set of metadata describing the group, a set of members in the groups, and a set of privileges assigned to users for protecting access to the group. Grid Grouper provides three mechanisms for adding members to a group: 1) directly adding a member 2) adding a subgroup to a group 3) making a group a composite of other groups. Directly adding a user as a member to a group is straight forward and these members are referred to as “Immediate Members”. Adding a subgroup to a group makes all the members of the subgroup members of the group in which the subgroup was added. Members in a group whose membership is granted by membership in a subgroup are referred to as “Effective Members”. A group can also be set up as a Composite group, which consists of a set operation (union,

intersection, or complement) on two other groups. For example a composite group consisting of the intersection of Group X and Group Y would contain all the members that are both members of Group X and Group Y. Members whose membership is granted through a composite group are referred to as “Composite Members”.

To protect access to groups in Grid Grouper, users can be assigned the following privileges on a group: View, Read, Update, Admin, Optin, and Optout. Users with the View privilege can see that the group exists. Users with the Read privilege can read basic information about the group. Users with the Update Privilege can manage memberships to the group as well as administer View, Read, and Update privileges. Users with the Admin privilege can modify/administer anything on the group: metadata, privileges, and memberships. Users with the Optin privilege can add themselves as a member to a group; similarly users with the Optout privilege can remove themselves from a group. By default Grid Grouper grants Read and View privileges to all users on each group.

Initially Grid Grouper has a root stem with one child stem named “Grouper Administration” (grouperadministration). The Grouper Administrative stem contains one group named “Grid Grouper Administrators” (grouperadministration:gridgrouperadministrators). The “Grid Grouper Administrators” is the super user group for Grid Grouper, all members of this group will have admin privileges on all the stems and groups within Grid Grouper. This group is initially empty, but at least one administrative user must be added during Grid Grouper installation. This can be done using the GridGrouperBootstrapper command line tool.

Grid Grouper Object Model

The Grid Grouper object model provides an API for applications and services to access groups managed by Grid Grouper. The object model can be used to enforce access control policies in applications. For example, the object model can be used for determining membership to a group in an application that allows access to a specific area of the application if the user is a member of a specified group. The Grouper object model can also be used to administrate Grid Grouper. As a testament to this, the Grid Grouper Admin UI application was built on top of the Grid Grouper object model. The Grid Grouper object model consists of several objects: GridGrouper, Stem, Group, Member, Membership, NamingPrivilege, and AccessPrivilege. The GridGrouper object corresponds to an instance of a Grid Grouper service; it provides high level operations such as finding stems and groups or determining whether or not a user is a member of a group, etc. The Stem object represents an instance of a stem within Grid Grouper. The Stem object provides operations for managing the stem: viewing metadata, managing child stems, managing child groups, managing stem privileges, etc. The Group object models a group instance within Grid Grouper, providing operations for managing metadata, managing privileges, and managing members. The remainder of this section will provide several code examples of performing common tasks with the Grid Grouper object model.

Determining if a Subject is a Member of a Group

```

try {
    String uri
    ="https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String user = "/O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    //Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri);

    //Determiner if the user is a member of the group.
    boolean isMember = grouper.isMemberOf(user, group);

    if(isMember){
        System.out.println("The user "+user+" is a member of "+group);
    }else{
        System.out.println("The user "+user+" is NOT a member of "+group);
    }
} catch (Exception e) {
    e.printStackTrace();
}

```

Listing All Members of a Group

```

try {
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri);

    // Obtain a handle to the group object.
    GroupI mygroup = grouper.findGroup(group);

    Set s = mygroup.getMembers();
    Iterator itr = s.iterator();
    //Iterate over and print out the members of the group
    while (itr.hasNext()) {
        Member m = (Member) itr.next();
        System.out.println("The user " + m.getSubjectId()
                           + " is a member of " + mygroup.getDisplayExtension());
    }
} catch (Exception e) {
    e.printStackTrace();
}

```

Adding a Member to a Group

```

try {
    //Group Administrators Grid Credentials
    GlobusCredential adminProxy = ProxyUtil.getDefaultProxy();
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String newMember = "/O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri,adminProxy);

    // Obtain a handle to the group object

```

```
GroupI mygroup = grouper.findGroup(group);

//Add the member to the group
mygroup.addMember(SubjectUtils.getSubject(newMember));

System.out.println("Successfully added the user " + newMember + " as a
member of the group " + group);
} catch (Exception e) {
    e.printStackTrace();
}
```

Removing a Member from a Group

```
try {
    //Group Administrators Grid Credentials
    GlobusCredential adminProxy = ProxyUtil.getDefaultProxy();
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String member = "/O=OSU/OU=BMI/OU=cagrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri,adminProxy);

    // Obtain a handle to the group object
    GroupI mygroup = grouper.findGroup(group);

    //Remove the member to the group
    mygroup.deleteMember(SubjectUtils.getSubject(member));

    System.out.println("Successfully removed the user " + member + "from the
group " + group);
} catch (Exception e) {
    e.printStackTrace();
}
```

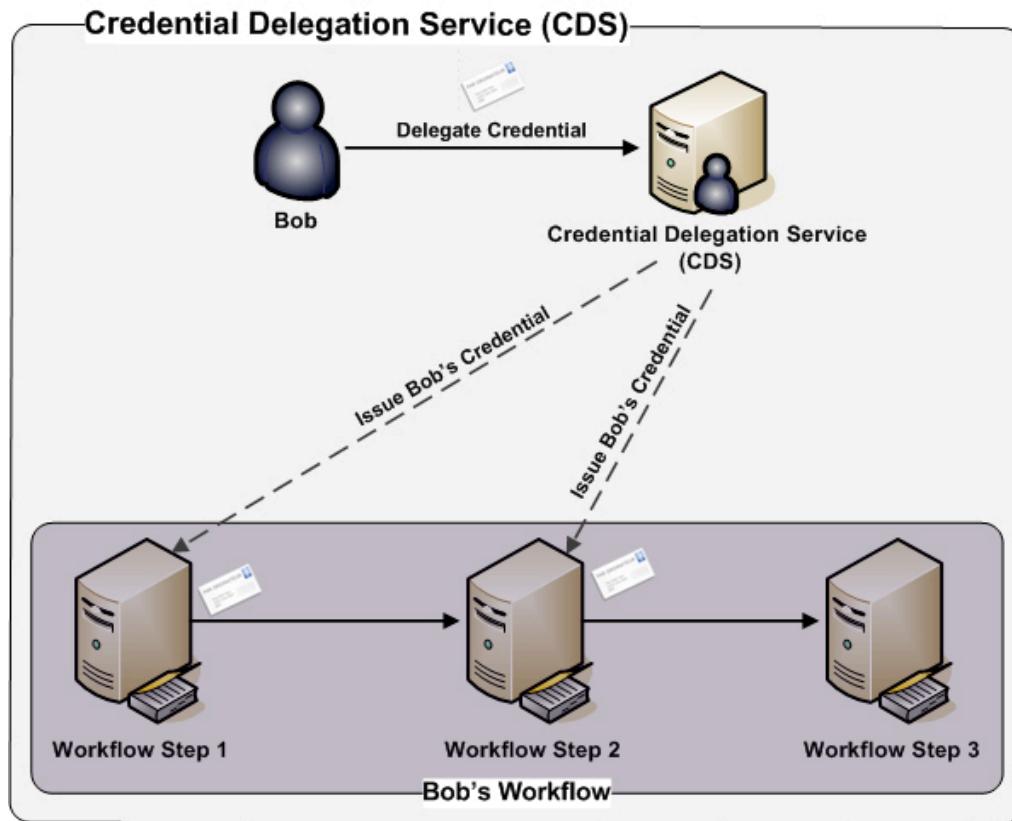
Credential Delegation Service (CDS) Overview

The Credential Delegation Service (CDS) is a WSRF-compliant Grid service that enables users/services (delegator) to delegate their Grid credentials to other users/services (delegatee) such that the delegatee(s) may act on the delegator's behalf. Consider a usecase where a user Bob wishes to invoke a simple workflow where three services interact with one another sequentially. Upon completion *Workflow Step 1* calls *Workflow Step 2*, which in turn calls *Workflow Step 3*. Now consider the following access control policies for these services:

- Bob has been granted access to each of these service resources
- Each of the services **does not** have access to the other services resources.

Since the services do not have access to one another's resources, the services must connect to one another as Bob in order to successfully execute this workflow. In order to interact with one another as Bob each of the first two service in the workflow: *Workflow Step 1* and *Workflow Step 2* must have Bob's Grid credentials such that they may authenticate as Bob. The CDS provides a secure mechanism for Bob to provide his credentials to the workflow service such that they may act on his behalf.

Besides enabling secure workflow the CDS enables secure execution of other technologies / frameworks including distributed queries and web application single sign on.



Programmatically Delegating a Credential (Identity Delegation Policy)

The example source code below illustrates how to delegate a credential to the Credential Delegation Service (CDS). In the example below the credential of the user whom is currently logged in or the default credential is delegated to the CDS for a period of four hours. The delegation policy specified in the example only allows the user, */O=caBIG/OU=caGrid/OU=Training/OU=Dorian/CN=jdoe* to get a credential. If the user, */O=caBIG/OU=caGrid/OU=Training/OU=Dorian/CN=jdoe* requests a credential, the CDS will issue them a credential for a time period of 1 hour. If needed the user may request additional credentials until the 4 hour delegation lifetime expires. Each additional credential will be valid for 1 hour or the time period left in the four hour delegation lifetime if that is less than 1 hour. Since the delegation path length of issued credentials is 0, the user, */O=caBIG/OU=caGrid/OU=Training/OU=Dorian/CN=jdoe* will not be allowed to delegate the credential to another party.

```
//The Service URL of the Credential Delegation Service (CDS)
String cdsURL =
"https://localhost:8443/wsrf/services/cagrid/CredentialDelegationService";

//The default credential or the user that is currently logged in.

GlobusCredential credential = ProxyUtil.getDefaultProxy();
```

```

//Specifies how long the delegation service can delegated this credential to
other parties.

ProxyLifetime delegationLifetime = new ProxyLifetime();
delegationLifetime.setHours(4);
delegationLifetime.setMinutes(0);
delegationLifetime.setSeconds(0);

//Specifies the path length of the credential being delegate the minumum is 1.

int delegationPathLength = 1;

//Specifies the how long credentials issued to allowed parties will be valid
for.

ProxyLifetime issuedCredentialLifetime = new ProxyLifetime();
issuedCredentialLifetime.setHours(1);
issuedCredentialLifetime.setMinutes(0);
issuedCredentialLifetime.setSeconds(0);

//Specifies the path length of the credentials issued to allowed parties. A
path length of 0 means that
//the requesting party cannot further delegate the credential.

int issuedCredentialPathLength = 0;

//Specifies the key length of the delegated credential

int keySize = ClientConstants.DEFAULT_KEY_SIZE;

//The policy stating which parties will be allowed to obtain a delegated
credential. The CDS will only
//issue credentials to parties listed in this policy.

List<String> parties = new ArrayList<String>();
parties.add("/O=caBIG/OU=caGrid/OU=Training/OU=Dorian/CN=jdoe");
IdentityDelegationPolicy policy =
Utils.createIdentityDelegationPolicy(parties);

//Create an instance of the delegation client, specifies the CDS Service URL
and the credential
//to be delegated.

DelegationUserClient client = new DelegationUserClient(cdsURL, credential);

//Delegates the credential and returns a reference which can later be used by
allowed parties to
//obtain a credential.

DelegatedCredentialReference ref =
client.delegateCredential(delegationLifetime, delegationPathLength, policy,
issuedCredentialLifetime, issuedCredentialPathLength, keySize);

```

Programmatically Delegating a Credential (Group Delegation Policy)

The example source code below illustrates how to delegate a credential to the Credential Delegation Service (CDS). In the example below the credential of the user whom is currently logged in or the default credential is delegated to the CDS for a period of four hours. The delegation policy specified in the example allows members of the *Training:Trainees* managed by the Grid Grouper <https://training03.cagrid.org:6443/wsrf/services/cagrid/GridGrouper>, to get a credential. If

a user that is a member of the *Training:Trainees* requests a credential, the CDS will issue them a credential for a time period of 1 hour. If needed the user may request additional credentials until the 4 hour delegation lifetime expires. Each additional credential will be valid for 1 hour or the time period left in the four hour delegation lifetime if that is less than 1 hour. Since the delegation path length of issued credentials is 0, any user whom are issued a credential will not be permitted to delegate the credential to another party.

```
//The Service URL of the Credential Delegation Service (CDS)

String cdsURL =
"https://localhost:8443/wsrf/services/cagrid/CredentialDelegationService";

//The default credential or the user that is currently logged in.

GlobusCredential credential = ProxyUtil.getDefaultProxy();

//Specifies how long the delegation service can delegated this credential to
other parties.

ProxyLifetime delegationLifetime = new ProxyLifetime();
delegationLifetime.setHours(4);
delegationLifetime.setMinutes(0);
delegationLifetime.setSeconds(0);

//Specifies the path length of the credential being delegate the minumum is 1.

int delegationPathLength = 1;

//Specifies the how long credentials issued to allowed parties will be valid
for.

ProxyLifetime issuedCredentialLifetime = new ProxyLifetime();
issuedCredentialLifetime.setHours(1);
issuedCredentialLifetime.setMinutes(0);
issuedCredentialLifetime.setSeconds(0);

//Specifies the path length of the credentials issued to allowed parties. A
path length of 0 means that
//the requesting party cannot further delegate the credential.

int issuedCredentialPathLength = 0;

//Specifies the key length of the delegated credential

int keySize = ClientConstants.DEFAULT_KEY_SIZE;

//The policy stating which parties will be allowed to obtain a delegated
credential. The CDS will only
//issue credentials to parties listed in this policy.

GroupDelegationPolicy policy = new GroupDelegationPolicy()
policy.setGridGrouperServiceURL("https://training03.cagrid.org:6443/wsrf/services/cagrid/GridGrouper");
policy.setGroupName("Training:Trainees");

//Create an instance of the delegation client, specifies the CDS Service URL
and the credential
//to be delegated.

DelegationUserClient client = new DelegationUserClient(cdsURL, credential);
```

```
//Delegates the credential and returns a reference which can later be used by
allowed parties to
//obtain a credential.

DelegatedCredentialReference ref =
client.delegateCredential(delegationLifetime, delegationPathLength, policy,
issuedCredentialLifetime, issuedCredentialPathLength, keySize);
```

Programmatically Obtaining a Delegated Credential

Obtaining a delegated credential allows one user (delegatee) to act on another user's behalf (delegator). In order to obtain a delegated credential the delegatee must have been granted access by the delegator to their credential. This is accomplished when the delegator delegates their credential to the CDS with a policy that grants the delegatee the right to obtain the credential. In order to obtain the delegator's credential the delegatee must (1) Specify which credential they wish to obtain and (2) authenticate with the CDS. A `DelegatedCredentialReference` is used to specify which credential a delegatee wishes to obtain. When a delegator delegates their credential they are given a `DelegatedCredentialReference` which they can give to the delegatee(s) such that they can later supply to the CDS to obtain a credential ([Guide to Delegating Credentials](#)). To authenticate with the CDS, the delegatee uses their own credential and assuming that the delegatee was granted access to by the delegator, the delegatee will be able to obtain a credential. The example source code below illustrates how to use the CDS client API to retrieve a delegated credential. For the purposes of the guide we assume that the `DelegatedCredentialReference` variable, `reference` was provided to us by a delegator.

```
//The default credential or the user that is currently logged in.

GlobusCredential credential = ProxyUtil.getDefaultProxy();

//A DelegatedCredentialReference is provided by the delegator to delegatee, it
//represents the delegated credential that the delegatee should obtain.

DelegatedCredentialReference reference =
getDelegatedCredentialReferenceFromDelegator();

//Create and instance of the delegate credential client, specifying the
//DelegatedCredentialReference and the credential of the delegatee. The
//DelegatedCredentialReference specifies which credential to obtain. The
//delegatee's credential is required to authenticate with the CDS such
//that the CDS may determine if the delegatee has been granted access
//to the credential in which they wish to obtain.

DelegatedCredentialUserClient client = new
DelegatedCredentialUserClient(reference, credential);

//The get credential method obtains a signed delegated credential from the CDS.

GlobusCredential delegatedCredential = client.getDelegatedCredential();

//Set the delegated credential as the default, the delegatee is now logged in
//as the delegator.

ProxyUtil.saveProxyAsDefault(delegatedCredential);
```

WebSSO (Web Single Sign On) Overview

Introduction

The WebSSO framework provides a comprehensive, Single Sign On solution for web application using caGrid's GAARDS Framework. Users may navigate between participating applications without being challenged to provide login credentials. Likewise, WebSSO establishes a grid session for the user, allowing them access to other grid services unchallenged.

Out of Box SSO Component

The WebSSO Team, along with the caGrid Architecture team, evaluated various open source technologies for selecting an out of box standard Web Single Sign On Product. JA-SIG's Central Authentication Service (CAS) was selected for these reasons:

- Provides a simple, spring-based, pluggable approach to add functionality
- Simple, non-intrusive, filter-based integration with target web applications (other ways possible too)
- Extendable Client/Server Protocol. Provides capability of returning signed SAML assertions back to client from the server, hence providing a higher level of security
- Extensive documentation, user base, and a larger development team.

JA-SIG's Central Authentication Service (CAS)

The caGrid's WebSSO Project uses CAS as the core component for providing the Single Sign On Framework. CAS is an authentication system originally created by Yale University to provide a trusted way for applications to authenticate a user. CAS became a JA-SIG project in December 2004.

CAS will be enhanced to use the caGrid's GAARDS framework in back-end to authenticate the user. Also this framework will be enhanced to obtain's user's grid credentials on the client side and provide it to the Target Web Application.

More details on CAS can be obtained from their website at: <http://www.ja-sig.org/products/cas/>.

It provides details about its inner-architecture, protocols, available features, and how to use them.

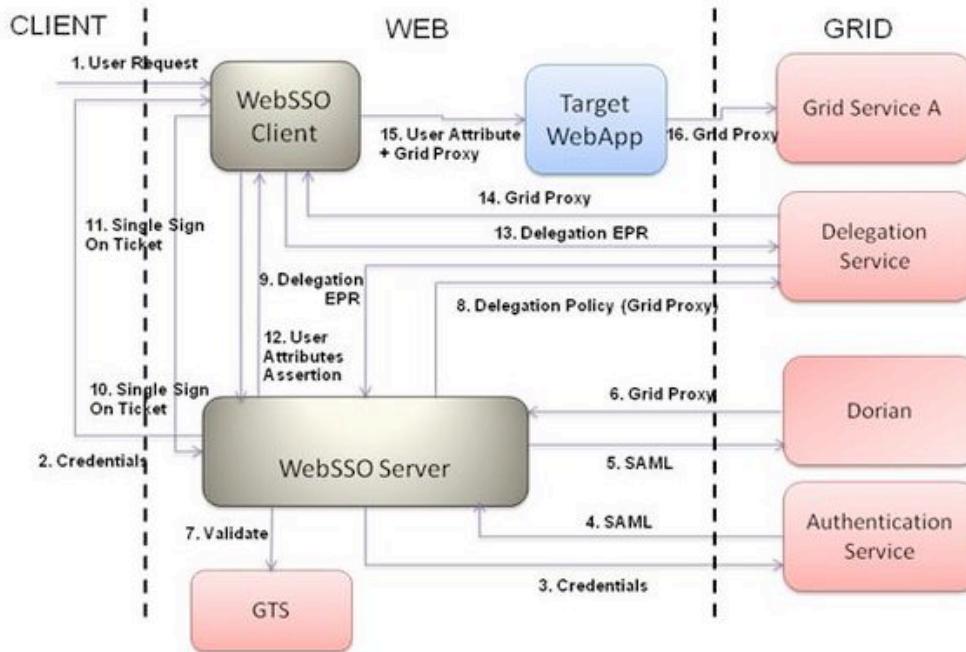
For the caGrid WebSSO Project, we employ the following JA-SIG CAS components:

- **CAS Server v3.1 Stable Release:** This is the server component of the SSO, which facilitates authentication of user's credentials, establishment of the Single Sign On Session, and granting the SSO Session Tickets.
- **JA-SIG's CAS Client for Java v3.0:** This is the client-side web agent that includes classes for ticket validation, proxy ticket acquisition, servlets, filters for implementing the client portion of the CAS protocol, and Assertion Java object for representing the results of a validation attempt. This library is usable for implementing custom CAS functionality, and can be used stand-alone to CASify existing applications.

WebSSO leverages the CAS Client for Java in such a way that it doesn't prevent other existing CAS clients from being used, since it doesn't modify the underlying client/server protocol or modify the client code in any way.

Overview of WebSSO Framework

The following diagram showcases all the components that are involved in the WebSSO framework. These components and their interactions are explained below.



Overview of WebSSO Workflow

WebSSO Framework Components

Following are the minimum set of components involved in an implementation of the WebSSO Framework. The diagram above depicts a single target application which is protected using WebSSO. If there is more than one application, then there will be a corresponding number of CAS Agents deployed along with them.

- **Client Browser**
The Client is the end user trying to access a Target Web Application using a browser. The client browser is useful for persisting the Single Sign On session using a cookie, thereby allowing the SSO Server to identify the user and not challenge him for credentials when redirected from another application.
- **Target Web Application**
This is the central authentication server which controls the entire Single Sign On framework. This component is provided by CAS. It also provides a login page which will be configured to adapt the look and feel of the CCTS Suite. It will be enhanced to communicate with caGrid's GAARDS
- **CAS Single Sign On Server**
This is the central authentication server which controls the entire Single Sign On

framework. This component is provided by CAS. It provides the login page which will be adapted to the look and feel of the CCTS Suite. It will be enhanced to employ caGrid's GAARDS framework to authenticate a user using their local credentials. In return, this server will obtain his grid credentials and pass them to the Credential Delegation Service and publish a delegation policy as in who all can access the grid credentials. Once user is authenticated, it generates a service ticket which is transferred back to the client. It then uses this ticket to confirm that the SSO Session has been established and retrieve user's attributes from the SSO Server.

- **CAS Agent**

This is the client component of the CAS Single Sign On framework which will be deployed and configured as part of the target web application. It is a standard client which all applications can integrate through the web.xml. It intercepts all the client's request to check if the single sign on session is established or not. If not it routes the user's request to the Central Sign On Server allowing the user to authenticate himself and establish the SSO Session. On return from the Central Server the client intercepts the user's request and validates the user's session against the server. On success it also interacts with the Delegation Service to obtain user's Grid Credential. Then CAS Agent forwards the user request along with the user's attributes and grid credentials to the target application.

- **Authentication Service**

To achieve federation, WebSSO allows a user to login using their local credentials. The CAS Single Sign On Server will be configured to point to an Authentication Service, which stands in front of a user's Identity Provider. This Authentication Service accepts their Username/Password credentials and authenticates against the configured Identity Provider. If successful, it retrieves the user's attributes from the Identity Provider and formulates a SAML Assertion which is sent back to the Central Server.

- **Dorian**

Once a user is authenticated from the Authentication service, the Central Server forwards the SAML Assertion to Dorian. The Central Server is configured to point to a single Dorian Server which acts as Identity Federation Service for the v0.5 Release. It generates their grid credentials (which is simply a short-lived proxy certificate) and returns the same to the Central Server.

- **Credential Delegation Service**

The Credential Delegation Service is being built by the caGrid team which will be used to store the User's Grid Credential along with a delegation policy. This delegation policy determines which applications have access to user's Grid Credentials. On successful obtaining the User's Grid Credential, the SSO Server publishes a delegation policy along with the User grid credentials to the Credential Delegation Service. On the client side the CAS Agent connects to the Credential Delegation Service using Target Application's Host Credentials and passes the Delegation EPR(End Point Reference) to retrieve User grid credentials. If it is permitted then the Credential Delegation Service returns the user's grid credentials back to the CAS Agent, which is then forwarded to the Target Application for it to use to access grid services.

- **GTS**

Both the WebSSO Server and Client use GTS (Grid Trust Fabric) to validate the

User's Grid Proxy . A Sync Description file is used to connect to the GTS Service running on the Grid. Both the WebSSO Server and Client use this file to connect to the GTS Server and validate if the user's proxy or CA has not been revoked. WebSSO provides facility to programmatically start SyncGTS both on client as well as server. If the user opts to start SyncGTS Daemon manually they can do so by following step at GTS page.

Single Sign in Workflow

Following is the process flow for components to establish a single sign on session:

1. User initiates a request to the target application (which is protected by the SSO Framework) by typing its URL into the browser. The user's request is intercepted by the CAS Agent, which searches for an established session. Since it does not exist, it redirects the user's request to the CAS Server providing the target application's URL as return point.
2. The CAS Server displays a login page, and the user provides their username/password pair for authentication.
3. The CAS server provides these credentials to the Authentication Service.
4. If the credentials are valid, the Authentication Service returns a signed SAML Assertion to the CAS Server.
5. The CAS server passes the signed SAML assertion to Dorian.
6. Dorian ensures that the SAML assertion is signed by a registered Authentication Service and returns short-term Grid Credentials (Grid Proxy) for the user.
7. They CAS server validates the Grid Proxy obtained from Dorian against the GTS to make sure that it is still valid and the CA has not be revoked.
8. Once validated, the CAS server obtains the list of Host Identities from the configuration files and formulates a delegation policy. It publishes this delegation policy, passing the user's Grid Credentials to the Credential Delegation Service
9. Once the Delegation Policy is published in the Credential Delegation Service successfully, it returns an End Point Reference (EPR) pointing to this Delegation Policy back to the WebSSO Server
10. Now it formulates a service ticket, attaches it as part of the URL as a GET parameter, and redirects the user back to the application using the provided return URL.
11. The CAS Agent for the application intercepts the request and retrieves the Service Ticket. It validates this Ticket with the CAS Server to ensure the user was authenticated recently.
12. CAS Server validates the ticket and returns an assertion object with the user's attributes to the client. The CAS Agent retrieves this information, and attaches it to user Request/Session.
13. The CAS Agent connects to the Credential Delegation Service using the Host Credentials of the Target Web Application to retrieve the User's Grid Credentials (Grid Proxy) by passing the Delegation EPR (End Point Reference).

14. The Delegation Server retrieves the Host Identity from the call and checks it against the user's published policy. If the application has been given delegation rights, it returns the Grid Proxy for that user back to the CAS Agent.
15. The CAS Agent validates the Grid Proxy against GTS and attaches it as an attribute to the Session and forwards the request to the target application.
16. The Target Application can retrieve all user attributes from the Request/Session including the user's Grid Credentials. It then can use this Grid Credential to access a grid service on user's behalf.

WebSSO Installation / Configuration

WebSSO has the following two components:

- **WebSSO Server**

The WebSSO server is installed centrally in a separate web container. It is responsible for establishing the Single Sign On Session. It interacts with Authentication Service (IdP) to validate user's local credentials and then interacts with Dorian (IFS) to obtain user's grid credentials. It also prepares a delegation policy describing which host identities can obtain user's grid credentials and publish it in the Central Delegation Service (CDS)

- **WebSSO Client**

The WebSSO Client is integrated with each target web application in the SSO realm. It is responsible for checking if a Single Sign On session has been established or not. If not, then route the user to WebSSO Server for authentication. Once authenticated and Single Sign On session established, the WebSSO client connects and retrieves user's attributes as well grid credential.

The next sections describe detailed steps on how to install and configure these components.

WebSSO Server Installation

This section contains information about installing and configuring Web Single Sign On (WebSSO) on a server.

Web Single Sign On (WebSSO) Server Installation And Configuration

The Web Single Sign On (WebSSO) is distributed both as a standalone project and a component other projects (such as caGrid). Each distribution contains a `websso` directory, herein referred to as WEBSSO_LOCATION. To install and configure the WEBSSO, please follow the steps below.

Note WebSSO Server release comes with its own patched version of JGlobus jar.

Step 1: Install Prerequisite Software

In order to install and run the WebSSO, the following pre-requisite software must be installed:

- [Java 1.5 JDK](#)
- [Ant 1.6.5](#)

- [Globus WS-Core with WS-Enum Support](#)
- [Tomcat 5.0.28](#)

Step 2: Setup environment variables

1. Create a `GLOBUS_LOCATION` environment variable and point it at the directory in which you installed Globus.
2. Create a `CATALINA_HOME` environment variable and point it at the directory in which you installed Tomcat.

Step 3: Obtain a Host Credential

The WebSSO must run as a secure service, so the hosting container must run with a host credential. A host credential consist of an X.509 certificate and private key. Dorian provides the ability to issue and manage host credentials. There are many methods of retrieving host credentials, including:

1. Requesting a credential from a known/trusted certificate authority (caGrid Certificate Authority). (**RECOMMENDED APPROACH**)
2. Standing up a Dorian service.
3. Standing up a simple certificate authority.

Step 4: Create Host Credential Keystore

1. Download and unzip gridca.
2. Bring up a command prompt and change to the `gridca` directory which you recently unzipped.
3. At the command line type `ant createTomcatKeystore`, this will execute a command line program which will guide you through creating the keystore.
4. In the *Enter a location and name for your keystore*: prompt enter a file name and location to create your keystore.
5. In the *Enter a password for your keystore*: prompt enter a password for your keystore.
6. In the *Enter the location of the certificate (PEM format)*: prompt enter the location of the host certificate you just created.
7. In the *Enter the location of the private key (PEM format)*: prompt enter the location of the private key you just created.
8. In the *Enter the current password of the private key*: prompt enter the password for your private key, if you private key does not have a password (most cases) just hit enter.
9. At this point the program will create your keystore at the location you specified. Below is a sample output of the program just described:

```

Enter a location and name for your keystore:mykeystore
Enter a password for your keystore:password
Enter the location of the certificate (PEM
format):w:\certificates\dwight.bmi.ohio-state.edu-cert.pem
Enter the location of the private key (PEM
format):w:\certificates\dwight.bmi.ohio-state.edu-key.pem
Enter the current password of the private key:

```

Step 5: Configure Globus To Trust the CA

- Note** This step is not necessary if you opt to start syncGTS programatically. For detailed steps refer Step 8 and Step 9.

We MUST configure Globus to trust the CA that issued the host credentials obtained in the previous step. To do this, place a copy of the certificate for the CA that issued the host credentials in the Globus trusted certificates directory. Unless otherwise specified during installation, this is usually `USER_HOME/.globus/certificates`. Globus requires all CA certificates in its trusted certificates directory to be in PEM format and to have a digit extension (0-9). For example, if a CA certificate is stored in the file `cacert.pem`, it should be copied to the directory `USER_HOME/.globus/certificates` (create directory if needed) with the file name `cacert.0`

Step 6: Configure Tomcat

Since the WebSSO Server would be running using SSL we need to configure Tomcat to enable SSL. To do so complete the following:

1. Edit the file `CATALINA_HOME/conf/server.xml` (example shown below).
2. Uncomment connector element for port 8443 (SSL)
3. Add a `keystoreFile` parameter containing the location of the keystore you just created.
4. Add a `keystorePass` parameter containing the the password of the keystore you just created.
5. Restart tomcat

```

<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<Connector port="8443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS"
keystoreFile="C:/Documents and Settings/user_account/mykeystore"
keystorePass="password" />

```

Step 7: Configure Tomcat Endorsed Jars

1. Delete jars from `CATALINA_HOME/common/endorsed`.
2. Download the `xercesImpl.jar` and place it in `CATALINA_HOME/common/endorsed`.

Step 8: Configure the WebSSO Server

The WebSSO is configured through `WEBSSO_LOCATION/src/resources/websso-properties.xml`. Below is an example of the `webssso` property file, followed by a description of each of the properties:

```

<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2005 rel. 3 U (http://www.altova.com) --
->
<websso-properties>
    <websso-server-information>
        <start-auto-syncgts>String</start-auto-syncgts>
        <trust-store-path>String</trust-store-path>
        <host-credential-certificate-file-path>String</host-credential-
certificate-file-path>
        <host-credential-key-file-path>String</host-credential-key-file-path>
    </websso-server-information>
    <credential-delegation-service-information>
        <service-url>http://www.altova.com</service-url>
        <delegation-lifetime-hours>0</delegation-lifetime-hours>
        <delegation-lifetime-minutes>0</delegation-lifetime-minutes>
        <delegation-lifetime-seconds>0</delegation-lifetime-seconds>
    </credential-delegation-service-information>
    <authentication-service-list>
        <authentication-service-information>
            <service-name>String</service-name>
            <service-url>http://www.altova.com</service-url>
            <dorian-information>
                <service-url>http://www.altova.com</service-url>
                <proxy-lifetime-hours>0</proxy-lifetime-hours>
                <proxy-lifetime-minutes>0</proxy-lifetime-minutes>
                <proxy-lifetime-seconds>0</proxy-lifetime-seconds>
                <proxy-delegation-path-length>0</proxy-delegation-path-length>
            </dorian-information>
        </authentication-service-information>
    </authentication-service-list>
    <delegated-applications-group>
        <group-name>String</group-name>
        <delegated-application-list>
            <delegated-application>
                <application-name>String</application-name>
                <host-identity>String</host-identity>
            </delegated-application>
        </delegated-application-list>
    </delegated-applications-group>
</websso-properties>

```

WebSSO Properties

- **websso-server-information** - This section contains information about the WebSSO Server.
 - **start-auto-syncgts** - This is a configuration parameter indicating whether the WebSSO Server should start SyncGTS automatically or not. "yes" indicates WebSSO Server to start the SyncGTS daemon
 - **trust-store-path** - This is the path to the trust store. This attribute is kept for future use.
- **host-credential-certificate-file-path** - This is the path to the WebSSO Server's Host Certificate File obtained in Step 3 above.
- **host-credential-key-file-path** - This is the path to the WebSSO Server's Host Key File obtained in Step 3 above.

- **credential-delegation-service-information** - This section is used to configure the Central Delegation Service, which will be used to publish the delegation policy for User's Grid Credentials.
 - **service-url** - This is the URL to the Central Delegation Service.
 - **delegation-lifetime-hours** - This is the hours for which the delegation policy remains alive.
 - **delegation-lifetime-minutes** - This is the minutes for which the delegation policy remains alive.
 - **delegation-lifetime-seconds** - This is the seconds for which the delegation policy remains alive.
- **authentication-service-list** - This is the list of all the Authentication Services which can be used to authenticate user's credentials. For CCTS Release there would be only one entry.
 - **authentication-service-information** - This is the entry for a single Authentication Service which consists of its name and URL.
 - **service-name** - This is the name of the Authentication Service.
 - **service-url** - This is the url where the Authentication Service is located.
 - **dorian-information** - This section is used to configure the Dorian Server which will be used to retrieve User's Grid Credentials
 - **service-url** - This is the URL to the Dorian Service.
 - **proxy-lifetime-hours** - This is the hours for which the proxy remains alive.
 - **proxy-lifetime-minutes** - This is the minutes for which the proxy remains alive.
 - **proxy-lifetime-seconds** - This is the seconds for which the proxy remains alive.
 - **proxy-delegation-path-length** - This is the delegation path length which should be set for proxy when created. It should be set to '0' thus avoiding any further delegation of proxy by the client.
- **delegated-applications-group** - These are the group of applications to which user's credentials are to be delegated. In future these will be provided as a choice to the user. As of now they are just static list.
 - **group-name** - This is the name given to the group of the applications to which a user's credentials is delegated.
 - **delegated-application-list** - These are the list of the applications to which user's credentials are to be delegated.
- **delegated-application** - This is the entry for an application to which the user's credentials are delegated..
 - **application-name** - This is the name of the application to which the user's credentials are delegated.

- **host-identity** - This is the host identity (obtained from the Host Credentials that are obtained from the Dorian) of the application to which the user's credentials are delegated.

Step 9: Copy the Sync Description File

In order to sync with the Grid Trust Fabric, the WebSSO Server needs a sync-description.xml file in its classpath to start SyncGTS programmatically if the `start-auto-syncgts` entry in WebSSO Configuration file is set to 'yes'. Depending upon the grid you are trying to connect to, you need to obtain the sync-description.xml file from the corresponding Grid Administrator. This file needs to be placed in `WEBSSO_LOCATION/src/resources` folder.

Note: If you have turned `start-auto-syncgts` to 'no', then the onus of syncing with the trust fabric relies on the administrator. This can be done manually by starting the syncGTS Daemon. For detailed steps, refer to GTS.

Step 10: Building WebSSO Server

If you have obtained a source release of the WebSSO, you will need to build it. Type the following from a command prompt:

```
%> cd WEBSSO_LOCATION  
%> ant clean all
```

Depending on the the WebSSO Server distribution, it may be required to build the entire project that the WebSSO Server is distributed with prior to building the WebSSO. For example, if you have obtained a caGrid source distribution, this is required. If you received a WebSSO Server standalone distribution, this is not required.

Step 11: Deploying WebSSO

Once you have configured a secure (Tomcat) container and built the WebSSO Application, you need to deploy the WebSSO to that container. To deploy the WebSSO to a secure Tomcat container, copy `cas.war` file from the `WEBSSO_LOCATION/build` folder into Tomcat's webapps folder.

WebSSO Client Integration

This section contains information about installing and configuring Web Single Sign On (WebSSO) on a client machine.

Web Single Sign Out Client (WebSSO Client) Installation And Configuration

The Web Single Sign Out Client (WebSSO Client) is distributed both as a standalone project and a component other projects (such as caGrid). Each distribution contains a `websso-client` directory, herein referred to as `WEBSSO_CLIENT_LOCATION`. To install and configure the WebSSO Client, please follow the steps below.

Note WebSSO Client release comes with its own patched version of JGlobus jar. So when integrating WebSSO Client into the Client Application use this patched JGlobus jar file.

Step 1: Install Prerequisite Software

In order to install and run the WebSSO, the following pre-requisite software must be installed:

- [Java 1.5 JDK](#)
- [Ant 1.6.5](#)
- [Globus WS-Core with WS-Enum Support](#)

The WebSSO Server has been configured and installed. The URL to this server would be used for configuring the WebSSO Client. Details on how to install and configure a WebSSO Server can be found at following location [WebSSO Administrators Guide](#)

Also, the target web application is a standard Java Web Application packaged in a war file. Details about a creating and packing a J2EE Web Application can be found on Sun's site at [Packaging Web Components](#).

Step 2: Building WebSSO Client

If you have obtained a source release of the WebSSO Client, you will need to build it.

Type the following from a command prompt:

```
%> cd WEBSSO_CLIENT_LOCATION
%> ant clean all
```

****Depending on the the WebSSO Client distribution, it may be required to build the entire project that the WebSSO Client is distributed with prior to building the WebSSO Client. For example, if you have obtained a caGrid source distribution, this is required. If you received a WebSSO Client standalone distribution, this is not required.****

Step 3: Obtain a Host Credential

The WebSSO Client provides capability to retrieve user's delegated credentials by connecting to the Credentials Delegation Service (CDS). To use this feature, a host credential needs to be obtained for the container/server hosting the web application which will integrate with WebSSO Client. A host credential consist of an X.509 certificate and private key. Dorian provides the ability to issue and manage host credentials. There are many methods of retrieving host credentials, including:

1. Requesting a credential from a known/trusted certificate authority (caGrid Certificate Authority).
2. Standing up a Dorian service.
3. Standing up a simple certificate authority.

Once a host credentials are obtained and stored on the server, the path to the certificate file and the key file should be noted. This will be used to configure the Delegation Lookup filter.

Step 4: Configure Server To Trust the Certificate Authority

Note: This step is not necessary if you opt to start syncGTS programmatically. For detailed steps refer Step 5 and Step 6.

To connect to the Credential Delegation Service (CDS), one must configure the server hosting the Web Application to trust the CA that issued the host credentials obtained in

the previous step. To do this, place a copy of the certificate for the CA that issued the host credentials in the Globus trusted certificates directory. Unless otherwise specified during installation, this is usually `USER_HOME/.globus/certificates`. Globus requires all CA certificates in its trusted certificates directory to be in PEM format and to have a digit extension (0-9). For example, if a CA certificate is stored in the file `cacert.pem`, it should be copied to the directory `USER_HOME/.globus/certificates` (create directory if needed) with the file name `cacert.0`

Step 5: Integrating the WebSSO Client

Once host credentials have been obtained, you can integrate the WebSSO Client into your Web Application. The WebSSO Client is released as a set of jar and configuration files. The jar files are found in the `WEBSSO_CLIENT_LOCATION/build/lib` folder and configuration files are in the `WEBSSO_CLIENT_LOCATION/build/conf` folder. All the jar files for the WebSSO Client should be added to the Web Application's library. This can be done by copying all the files from the `WEBSSO_CLIENT_LOCATION/build/lib` into your `WEBAPP_LOCATION/WEB-INF/lib`.

The `WEBSSO_CLIENT_LOCATION/build/conf` contains the following three configuration files:

- `web.xml`
- `cas-client.properties`
- `securityConfiguration.xml`

The **securityConfiguration.xml** file is a spring configuration files which contains the configuration setting for CAS Filters which are packaged as part of WebSSO Client. This file should be copied as it is in the `WEBAPP_LOCATION/WEB-INF/` folder.

The **cas-client.properties** is the file which contains configuration for connecting to the Central CAS Single Sign On Server. This file should be copied after modifications in the `WEBAPP_LOCATION/WEB-INF/` folder.

The **web.xml** is a sample file provided to show entries which needs to be made to your `web.xml` file to enable the WebSSO Client capabilities in your application.

Step 5.1: Configuring the cas-client.properties file

The cas client is configured through a properties file:

`WEBSSO_CLIENT_LOCATION/build/conf/cas-client.properties`. Below is an template of the `cas-client.properties` file, followed by a description of each of the properties:

```
cas.server.gateway=false
cas.server.renew=false
cas.server.url=https://<<CAS_SERVER_ADDRESS>>:<<CAS_SERVER_PORT>>/cas/
cas.server.proxyCallbackUrl=https://<<CAS_SERVER_ADDRESS>>:<<CAS_SERVER_PORT>>/manager/proxy/Receptor
cas.client.serverName=<<WEBAPP_ADDRESS>>:<<WEBAPP_PORT>>
cas.client.proxyCallbackUrl=https://<<WEBAPP_ADDRESS>>:<<WEBAPP_PORT>>/myClientApp/casProxyReceptor
```

cas-client.properties

- **cas.server.gateway** - This is a property indicating whether the login screen should be displayed to the user or not. This option should be always set to 'false'.

- **cas.server.renew** - This is a property if set to true would require the user to login again irrespective of whether the Single Sign On session has been established or not. For general use, this should be set to 'false'.
- **cas.server.url** - This is the URL to the WebSSO's CAS Server.
Note The CAS Server is generally installed in a secured container as a result of which the URL should have 'https'.
- **cas.server.proxyCallbackUrl** - This is the server proxy callback URL. This URL should point to the CAS Server/Port.
Note The feature that uses this URL is not used in WebSSO's implementation of CAS.
- **cas.client.serverName** - This is the client server address. It consist of the fully qualified server name and the port at which the application is running.
Note This port should be the SSL port if you are running secured application.
- **cas.client.proxyCallbackUrl** - This is the client proxy callback URL. This URL should point to the server and the port where the client application is running.
Note This port should be the SSL port if you are running secured application.

An example file follows:

```
cas.server.gateway=false
cas.server.renew=false
cas.server.url=https://localhosta:8443/cas/
cas.server.proxyCallbackUrl=https://localhosta:8443/manager/proxy/Receptor
cas.client.serverName=localhostb:8443
cas.client.proxyCallbackUrl=https://localhostb:8443/myClientApp/casProxyRecepto
r
```

Here **localhosta** is hosting the CAS Central Single Sign On Server and **localhostb** is hosting client application.

Note: Also since both the applications are assumed to be running securely hence the default SSL ports for tomcats are shown.

This file should be placed in the `WEB_LOCATION/WEB-INF/` folder of the client web application.

Step 5.2: Modifying the `web.xml` file

Once the `cas-client.properties` file is configured, you need to enable the WebSSO Client filters. To do this, make an entry into the `web.xml`. A sample `web.xml` is show below. This file can be found in the `WEBSSO_CLIENT_LOCATION/build/conf/` location of the WebSSO Client Release:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <servlet>
    <servlet-name>Start Auto Sync GTS</servlet-name>
```

```

<servlet-class>
    org.cagrid.gaards.websso.client.utils.StartSyncGTSServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>test.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/securityConfiguration.xml
    </param-value>
</context-param>

<filter>
    <filter-name>CAS Authentication Filter</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetBeanName</param-name>
        <param-value>casAuthenticationFilter</param-value>
    </init-param>
</filter>

<filter>
    <filter-name>CAS Validation Filter</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetBeanName</param-name>
        <param-value>casValidationFilter</param-value>
    </init-param>
</filter>

<filter>
    <filter-name>caGRID WebSSO Attribute Loader Filter</filter-name>
    <filter-
class>org.cagrid.gaards.websso.client.filter.CaGridWebSSOAttributeLoaderFilter</filter-class>
    </filter>

<filter>
    <filter-name>caGRID WebSSO Delegation Lookup Filter</filter-name>
    <filter-
class>org.cagrid.gaards.websso.client.filter.CaGridWebSSODelegationLookupFilter</filter-class>
    <init-param>
        <param-name>certificate-file-path</param-name>
        <param-value>C:/Documents and
Settings/MODI/.cagrid/certificates/MODIK_LAP-cert.pem</param-value>
    </init-param>
    <init-param>
        <param-name>key-file-path</param-name>

```

```

        <param-value>C:/Documents and
Settings/MODI/.cagrid/certificates/MODIK_LAP-key.pem</param-value>
    </init-param>
</filter>

<filter>
    <filter-name>caGRID WebSSO Logout Filter</filter-name>
    <filter-
class>org.cagrid.gaards.websso.client.filter.CaGridWebSSOLogoutFilter</filter-
class>
    <init-param>
        <param-name>logout-landing-url</param-name>
        <param-value>https://localhost:58443/sample/hello</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>CAS Validation Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>caGRID WebSSO Attribute Loader Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>caGRID WebSSO Delegation Lookup Filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>caGRID WebSSO Logout Filter</filter-name>
    <url-pattern>/logout/*</url-pattern>
</filter-mapping>

<listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

Following are the entries in the web.xml file which should be added to the Client Web Application's web.xml file:

Start-up Entries

- **Start Auto Sync GTS** - This entry is to load the `org.cagrid.gaards.websso.client.utils.StartSyncGTSServlet` Servlet on start up. This Servlet uses the sync-description.xml file from the classpath and starts the SyncGTS Daemon. If the target application doesn't want to automatically start the SyncGTS Daemon then they should remove this entry from the web.xml.

Note: Depending how many start up servlets are configured for your application you would need to give appropriate number (based on start up order) to the **load-on-startup** entry

Other Entries

- **contextConfigLocation** - This entry is to define the spring based `securityConfiguration.xml` file entry which is needed by the CAS Client. This entry should be copied as it is in the client web application's `web.xml` file
- **listener-class** - This entry points to the listener class which should be used.

Filter Entries

- **CAS Authentication Filter** - This is the entry for the CAS Authentication Filter. This filter checks if the User's Single Sign On Session has been established or not. This entry should be copied as it is in the client web application's `web.xml` file
- **CAS Validation Filter** - This is the entry for the CAS Validation Filter. This filter validates the User's Authentication Session and then retrieve User's Attributes from the Central CAS Single Sign On Server. This entry should be copied as it is in the client web application's `web.xml` file
- **caGRID WebSSO Attribute Loader Filter** - This is the entry for the caGRID WebSSO Attribute Loader Filter. This filter reads the attributes which are retrieved from the Central CAS Single Sign On Server and loads each of them into the HTTP Session. This entry should be copied as it is in the client web application's `web.xml` file
- **caGRID WebSSO Delegation Lookup Filter** - This is the entry for the caGRID WebSSO Delegation Lookup Filter. This filter connects to the Credential Delegation Service (CDS) using the Client Application's Host Credentials and retrieves the User's Grid Credentials. It then stores the retrieved User's Grid Credentials into the session as a session attribute. This filter need two parameter pointing to the certificate and the key file for the Client Application's Host Credential.
 - **certificate-file-path** - points to the certificate file for the Client Application's Host Credentials.
 - **key-file-path** - points to the key file for the Client Application's Host Credentials.
- **caGRID WebSSO Logout Filter** - This is the entry for the caGRID WebSSO Logout Filter. This filter connects to the WebSSO Server and issues a logout call. On receiving this logout the CAS Server terminates the SSO Session and issues a call to the Credential Delegation Service (CDS) to destroy user's delegation policy.
 - **logout-landing-url** - points to the location of the target webapplication where the user should be rerouted by the WebSSO Server upon successful logout.

Note These filters have to be defined in the above mentioned order as each is dependent on previous filter.

Once these filters are defined, a mapping must be provided for them. In the sample shown, it is mapped to protect everything within the web context root of the application. If you want to protect anything within a specific context URL, you need to provide that entry.

Step 6: Copying Sync Description File

In order to sync with the Grid Trust Fabric, the WebSSO Server needs a sync-description.xml file in its classpath to start SyncGTS programatically if the **Start Auto Sync GTS** Servlet is configured in the web.xml file above. Depending upon the grid you are trying to connect to, you need to obtain the sync-description.xml file from the corresponding Grid Administrator. This file needs to be placed in the classpath within the target application. This can be done by placing the file in the classes folder of the target application web archive (war file).

Note: if you haven't configure the **Start Auto Sync GTS** start up servlet, then the onus of syncing with the trust fabric relies on the administrator. This can be done manually by starting the syncGTS Daemon on the target web application. For detailed steps refer to GTS

Step 7: Establish trust with WebSSO CAS Server

In order for the client to trust the WebSSO Server you need to install the WebSSO Server's Public CA Cert into its truststore. This can be done by the following steps

1. On the target application server, simply copy the cacerts to cacerts.old (to save it just in case)
2. Run the java program given below by giving the following command - `run java InstallCert <><>:<><>` (i.e. provide the argument "`<><>:<><>`" to the executable "InstallCert"). (adapted source code for InstallCert.java from Sun blog by Andreas Sterbenz is shown below)
3. Answer 1 to the prompt.

```
/*
 * @(#)InstallCert.java 1.1 06/10/09
 *
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * Use is subject to license terms.
 */

import java.io.*;
import java.net.URL;

import java.security.*;
import java.security.cert.*;

import javax.net.ssl.*;

public class InstallCert {

    public static void main(String[] args) throws Exception {
        String host;
        int port;
```

```
char[] passphrase;
if ((args.length == 1) || (args.length == 2)) {
    String[] c = args[0].split(":");
    host = c[0];
    port = (c.length == 1) ? 443 : Integer.parseInt(c[1]);
    String p = (args.length == 1) ? "changeit" : args[1];
    passphrase = p.toCharArray();
} else {
    System.out.println("Usage: java InstallCert <host>[:port]
[passphrase]");
    return;
}

File file = new File("jssecacerts");
if (file.isFile() == false) {
    char SEP = File.separatorChar;
    File dir = new File(System.getProperty("java.home") + SEP
        + "lib" + SEP + "security");
    file = new File(dir, "jssecacerts");
    if (file.isFile() == false) {
        file = new File(dir, "cacerts");
    }
}
System.out.println("Loading KeyStore " + file + "...");
InputStream in = new FileInputStream(file);
KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
ks.load(in, passphrase);
in.close();

SSLContext context = SSLContext.getInstance("TLS");
TrustManagerFactory tmf =

TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(ks);
X509TrustManager defaultTrustManager =
(X509TrustManager)tmf.getTrustManagers()[0];
SavingTrustManager tm = new SavingTrustManager(defaultTrustManager);
context.init(null, new TrustManager[] {tm}, null);
SSLSocketFactory factory = context.getSocketFactory();

System.out.println("Opening connection to " + host + ":" + port + "...");
SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
socket.setSoTimeout(10000);
try {
    System.out.println("Starting SSL handshake...");
    socket.startHandshake();
    socket.close();
    System.out.println();
    System.out.println("No errors, certificate is already trusted");
} catch (SSLEException e) {
    System.out.println();
    e.printStackTrace(System.out);
}

X509Certificate[] chain = tm.chain;
if (chain == null) {
    System.out.println("Could not obtain server certificate chain");
    return;
}

BufferedReader reader =
    new BufferedReader(new InputStreamReader(System.in));
```

```

System.out.println();
System.out.println("Server sent " + chain.length + " certificate(s):");
System.out.println();
MessageDigest sha1 = MessageDigest.getInstance("SHA1");
MessageDigest md5 = MessageDigest.getInstance("MD5");
for (int i = 0; i < chain.length; i++) {
    X509Certificate cert = chain[i];
    System.out.println
        (" " + (i + 1) + " Subject " + cert.getSubjectDN());
    System.out.println("  Issuer " + cert.getIssuerDN());
    sha1.update(cert.getEncoded());
    System.out.println("    sha1    " + toHexString(sha1.digest()));
    md5.update(cert.getEncoded());
    System.out.println("    md5    " + toHexString(md5.digest()));
    System.out.println();
}
System.out.println("Enter certificate to add to trusted keystore or 'q' to
quit: [1]");
String line = reader.readLine().trim();
int k;
try {
    k = (line.length() == 0) ? 0 : Integer.parseInt(line) - 1;
} catch (NumberFormatException e) {
    System.out.println("KeyStore not changed");
    return;
}

X509Certificate cert = chain[k];
String alias = host + "-" + (k + 1);
ks.setCertificateEntry(alias, cert);

OutputStream out = new FileOutputStream(file);
ks.store(out, passphrase);
out.close();

System.out.println();
System.out.println(cert);
System.out.println();
System.out.println
    ("Added certificate to keystore 'cacerts' using alias '"
    + alias + "'");
}

private static final char[] HEXDIGITS = "0123456789abcdef".toCharArray();

private static String toHexString(byte[] bytes) {
StringBuilder sb = new StringBuilder(bytes.length * 3);
for (int b : bytes) {
    b &= 0xff;
    sb.append(HEXDIGITS[b >> 4]);
    sb.append(HEXDIGITS[b & 15]);
    sb.append(' ');
}
return sb.toString();
}

private static class SavingTrustManager implements X509TrustManager {

    private final X509TrustManager tm;
    private X509Certificate[] chain;

    SavingTrustManager(X509TrustManager tm) {

```

```

        this.tm = tm;
    }

    public X509Certificate[] getAcceptedIssuers() {
        throw new UnsupportedOperationException();
    }

    public void checkClientTrusted(X509Certificate[] chain, String authType)
        throws CertificateException {
        throw new UnsupportedOperationException();
    }

    public void checkServerTrusted(X509Certificate[] chain, String authType)
        throws CertificateException {
        this.chain = chain;
        tm.checkServerTrusted(chain, authType);
    }
}

```

WebSSO Client APIs

Retrieving User Attributes Inside Client Application

Once the WebSSO Client has been configured, you need to program your application to retrieve the session attributes which are injected by the WebSSO Client. Following is the list of session attributes which hold values of the corresponding User Attributes:

Session Attribute Name	Information
CAGRID_SSO_GRID_IDENTITY	User's Grid Identity
CAGRID_SSO_FIRST_NAME	User's First Name
CAGRID_SSO_LAST_NAME	User's Last Name
CAGRID_SSO_EMAIL_ID	User's Email Id
CAGRID_SSO_DELEGATION_SERVICE_EPR	Delegation Service's End Point Reference
CAGRID_SSO_GRID_CREDENTIAL	User's Grid Credential

These attributes can be retrieved from the HTTP session programmatically by specifying the individual session attribute name. Below is an example servlet which retrieves all the attribute and prints them on the screen.

```

package test;

import java.io.*;
import java.util.Enumeration;

import javax.servlet.http.*;
import javax.servlet.*;

public class HelloServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Returns a writer to write to the browser
        PrintWriter out = response.getWriter();

```

```
// Writes the string to the browser.  
out.println("Begin Printing User Attributes");  
Enumeration attributeNames = request.getSession().getAttributeNames();  
for (; attributeNames.hasMoreElements(); ) {  
    // Get the name of the attribute  
    String name = (String)attributeNames.nextElement();  
    if (name.contains("CAGRID"))  
    {  
        Object value = (Object)request.getSession().getAttribute(name);  
        out.println("<<<< ATTRIBUTE NAME >>>>" + name + "<<<< ATTRIBUTE  
VALUE >>>>" + value);  
    }  
}  
out.println("Done Printing User Attributes");  
out.close();  
}  
}
```


Chapter 5 caGrid Service Developers

Introduce Service Developers

Deserialization and Serialization Examples

Introduce comes with helper tools for serializing and deserializing objects.

Deserialization

The example below if the the cagrid transfer service. This example is deserializing an xml file into a `TransferServiceContextResourceProperties` object.

```
TransferServiceContextResourceProperties props = null;
try {
    props = (TransferServiceContextResourceProperties)
    Utils.deserializeObject(new FileReader(persistenceDir
        + File.separator + requestedID + ".xml"),
    TransferServiceContextResourceProperties.class);
} catch (Exception e) {
    logger.info("Cannot find or deserialize the resource properties
describing this transfer object: "
    + requestedID);
    e.printStackTrace();
}
```

Serialization

Write the object out that we serialized above. This will use the QName of the schema element that this object adheres to and write the object out to the `test.xml` file.

```
try {
    Utils.serializeDocument("test.xml", props, new
    QName("http://transfer.caGrid.org/TransferService/Context",
"TransferServiceContextResourceProperties"));
} catch (Exception e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
```

caGrid Debugging Information

If you are using the Globus container, make sure you pass the `-debug` flag when you start the container.

Logging Information

Use log4j in Tomcat

```
cp $GLOBUS_LOCATION/lib/log4j-1.2.8.jar $CATALINA_HOME/webapps/wsrf/WEB-INF/lib
cp $GLOBUS_LOCATION/lib/commons-logging.jar $CATALINA_HOME/webapps/wsrf/WEB-
INF/lib
```

edit \$CATALINA_HOME/webapps/wsrf/WEB-INF/classes/log4j.properties as you see fit.

Use log4j on the client

To use log4j logging on the client, you will need to configure your client to load a log4j configuration file. You can find details (probably more than you want) on the variety of ways to configure log4j in its manual. Probably the simplest way is to just make sure your configuration file is named log4j.properties, and place it somewhere on your classpath (for example in one of your jars). If you are using Introduce, you can just drop the file in your src folder and rebuild.

Turn on caGrid debugging log messages

Edit your log4j.properties (see above) and add this line:

```
log4j.category.gov.nih.nci.cagrid=DEBUG
```

For problems with advertisement/registration to the index service, add these lines as well:

```
log4j.category.org.globus.mds=DEBUG  
log4j.category.org.globus.wsrf.impl.servicegroup=DEBUG
```

Turn on Message Level SOAP logging (slow!)

```
cp $GLOBUS_LOCATION/lib/log4j-1.2.8.jar $CATALINA_HOME/webapps/wsrf/WEB-INF/lib  
cp $GLOBUS_LOCATION/lib/commons-logging.jar $CATALINA_HOME/webapps/wsrf/WEB-INF/lib
```

edit \$CATALINA_HOME/webapps/wsrf/WEB-INF/classes/log4j.properties replace this line:

```
# log4j.category.org.globus.wsrf.handlers.MessageLoggingHandler=DEBUG
```

with this:

```
log4j.category.org.globus.wsrf.handlers.MessageLoggingHandler=DEBUG
```

Debug Services running in Tomcat with Eclipse

When you start tomcat, use:

```
$CATALINA_HOME\bin\catalina.sh jpda start
```

You may need to edit catalina.sh (or bat) in windows to have these properties:

```
set JPDA_ADDRESS=8000  
set JPDA_TRANSPORT=dt_socket
```

In Eclipse, go to the Debug window and select new "Remote Java Application," select your service's project, and use the rest of the defaults. You should now be able to set breakpoints in your service's source code and debug it running in Tomcat.

SOAP Viewing

<http://docs.pushtotest.com/axisdocs/user-guide.html#tcpmon>

Debug ant tasks spawned by an Introduce build process

- Set a breakpoint in your custom Ant task code.
- Open the launch configuration dialog on the Ant config for the build to be debugged

- Go to the JRE tab
- In the VM arguments specify for the VM to launch to enable remote debug:

```
"-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000"
```

- The Ant build will launch and wait for a remote debug attach
- Top level **Run menu > Debug**
- Create a new Remote Java Application launch configuration
- Set the source lookup path as required for your code
- Debug

You should hit your breakpoint and be able to step through the code to determine what is the root problem.

Launch the Ant build in run mode.

Using the Client

Introduce generates a client API for the service which is exactly as described within the graphical editing environment (see Auto-boxing/Unboxing or Service Operations). This client API can be used in order to leverage this type of service from another application or service. The API contains four constructors which can be used, each of which is different depending on having a handle or just an address and the need for security to be used.

```
/**
 * Takes in the url of the service to connect to as a string
 */
HelloWorldClient(String url)

/**
 * Takes in the url of the service to connect to as a string and
 * a proxy to be used to represent the credentials or the caller
 */
HelloWorldClient(String url, GlobusCredential proxy)

/**
 * Takes in the epr which refers to the service or resource
 */
HelloWorldClient(EndpointReferenceType epr)

/**
 * Takes in the epr which refers to the service or resource and
 * a proxy to be used to represent the credentials or the caller
 */
HelloWorldClient(EndpointReferenceType epr, GlobusCredential proxy)
```

Once a client handle is constructed each of the operations which were created in the service are available as operations to this newly constructed client instance. Below is an example snippet of code which creates a new client handle to a service called "HelloWorld" and calls the "echo" operation.

```
try {
    HelloWorldClient client = new
    HelloWorldClient("http://localhost:8080/wsrf/services/HelloWorld");
    client.echo("Testing");
```

```

} catch (Exception e) {
    System.out.println("Problem creating handle to or calling service" +
e.getMessage(), e);
}

```

Advanced Client Usage

Printing detailed faults

The example below shows how a client can print a more detailed stack trace from an exception that is received from the service.

```

try {
    ...
} catch (Exception e) {
    gov.nih.nci.cagrid.common.FaultUtil.printFault(e);
}

```

Using Subscription and Notification

If the Introduce service you are using supports the Notification resource framework option than the client will be enabled for subscribing to that service and listening for particular changes to resource properties. An example below shows how to perform the subscription:

```

IntroduceTestNotificationServiceClient client = new
IntroduceTestNotificationServiceClient("http://localhost:8080/wsrf/services/Int
roduceTestNotificationServiceClient");
client.subscribe(new QName("http://testing.org","TestResourceProperty"));

```

In the above code we have an example client that is subscribing to a service to listen for changes to the TestResourceProperty resource property of the service. If the value of this resource property gets changed in the resource a notification will be sent out to all registered subscribers. Now we need to implement some code so that we receive these notifications. The introduce client generated for this service will automatically be able to listen for these changes. For us to be able to perform our own actions when we receive a notification we must overload the deliver operation in our client:

```

public void deliver(List topicPath, EndpointReferenceType producer, Object
message) {
    org.oasis.wsrf.properties.ResourcePropertyValueChangeNotificationType
changeMessage =
((org.globus.wsrf.core.notification.ResourcePropertyValueChangeNotificationElem
entType) message)
    .getResourcePropertyValueChangeNotification();

    if (changeMessage != null) {
        receivedNotificationCount++;
        try {
            System.out.println("GOT NOTIFICATION: " +
changeMessage.getNewValue().get_any()[0].getAsString());
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

Loading a proxy from a file

If you want to use a specific proxy file, other than the locations that Globus will automatically look. The ProxyUtil class can be used to load the proxy and create a GlobusCredential.

```
GlobusCredential creds = null;
try {
    creds =
gov.nih.nci.cagrid.common.security.ProxyUtil.loadProxy("user.proxy");
    System.out.println("Using proxy with id= " + creds.getIdentity() + " and
lifetime "
                    + creds.getTimeLeft());
} catch (Exception e1) {
    System.out.println("No proxy file loaded so running with no
credentials");
}
```

Implementing the Service

When an operation is added in the Introduce GDE and the save button is clicked.

Introduce will add the new stubbed method into the <service

package>.service.<service name>Impl.java class. The developer is then responsible for implementing the method prior to deployment. For example the snippet below would be generated in the <service package>.service.<service name>Impl.java if the developer added an *add* operation to the service with Introduce that took in two integers and returned an integer.

```
public int add(java.math.BigInteger a,java.math.BigInteger b) throws
RemoteException {
    //TODO: Implement this autogenerated method
    throw new RemoteException("Not yet implemented");
}
```

The developer would then have to edit this method to implement the logic the service would execute on invocation.

```
public int add(java.math.BigInteger a,java.math.BigInteger b) throws
RemoteException {
    return a + b;
}
```

Advanced Service Development Topic

Obtaining the caller's identity

In the service it is sometime required to get the identity of the caller. The identity is carried in the context of the message which was passed into the service. In order to get this information the following lines will be needed.

```
String userDN =
gov.nih.nci.cagrid.introduce.servicetools.security.SecurityUtils.getCallerIdent
ity()
```

Accessing service properties

Service properties are the properties that the main service can be configured with that all service contexts deployed with that service can access. These properties are configured at deploy time and will likely need to be accessed in the service. Below is example code for how to access these properties where the main service was named **HelloWorld** and the name of the service property was **testing**.

```
HelloWorldConfiguration.getConfiguration().getTesting();
```

Getting a Handle to a Resource

When developing in the service context you must remember that the ServiceImpl class is just the service interface but the service should act on the addressed resource as there may be more than one resource instance being used by the service at a time. An example of how to get a handle to the resource which is being addressed by the current caller (example from the counter service context as described in the section on Utilizing the factory pattern):

```
public void incrementCounter() throws RemoteException {
    int currentCounter;
    try {
        currentCounter = getResourceHome().getAddressedResource().getIntValue();

        currentCounter++;
        getResourceHome().getAddressedResource().setIntValue(currentCounter);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Programmatically accessing and/or editing resource properties (metadata)

From the resource there will be getters and setters automatically generated for the resource properties of your service. For instance, if you look at the above example, you can see that the counter service has an "int" resource property. This enables introduce to automatically generated getters and setters for that resource property that the service implementer can use to access and modify values in the resource properties.

Creating a Security Descriptor for a Resource

Sometimes it is necessary to set a security descriptor on an instance of a resource. This might be to protect resource instances so that only the creator of the resource can use it. In order to do this you have to make sure that when the context was generated in introduce that you had selected the "secure" resource framework option. This will enable the resource to have a `setSecurityDescriptor()` method enabled on it. When the resource is being created, as in the example below, the security descriptor should be generated and set on the resource. Once set it will be utilized on any access to the resource. Below is an example method from the counter service used in the utilizing a factory pattern section. This is the method that Introduce generated to create the counter instance for the caller. You can see in this code that there is a place documenting how to set the security descriptor on the resource.

```
public example.counter.context.stubs.types.CounterContextReference
createCounter() throws RemoteException {
```

```

        org.apache.axis.message.addressing.EndpointReferenceType epr = new
        org.apache.axis.message.addressing.EndpointReferenceType();
        example.counter.context.service.globus.resource.BaseResourceHome home =
null;
        org.globus.wsrf.ResourceKey resourceKey = null;
        org.apache.axis.MessageContext ctx =
org.apache.axis.MessageContext.getCurrentContext();
        String servicePath = ctx.getTargetService();
        String homeName = org.globus.wsrf.Constants.JNDI_SERVICES_BASE_NAME +
servicePath + "/" + "counterContextHome";

        try {
            javax.naming.Context initialContext = new
            javax.naming.InitialContext();
            home =
(example.counter.context.service.globus.resource.BaseResourceHome)
initialContext.lookup(homeName);
            resourceKey = home.createResource();

            // Grab the newly created resource

            example.counter.context.service.globus.resource.CounterContextResource
thisResource =
(example.counter.context.service.globus.resource.CounterContextResource)home.fi-
nd(resourceKey);

            // This is where the creator of this resource type can set whatever
needs
            // to be set on the resource so that it can function appropriately for
instance
            // if you want the resource to only have the query string then there
is where you would
            // give it the query string.

            // sample of setting creator only security. This will only allow the
caller that created
            // this resource to be able to use it.

//thisResource.setSecurityDescriptor(gov.nih.nci.cagrid.introduce.servicetools.
security.SecurityUtils.createCreatorOnlyResourceSecurityDescriptor());



        String transportURL = (String)
ctx.getProperty(org.apache.axis.MessageContext.TRANS_URL);
        transportURL = transportURL.substring(0,transportURL.lastIndexOf('/')+
1 );
        transportURL += "CounterContext";
        epr =
org.globus.wsrf.utils.AddressingUtils.createEndpointReference(transportURL,reso-
urceKey);
        } catch (Exception e) {
            throw new RemoteException("Error looking up CounterContext home:" +
e.getMessage(), e);
        }

        //return the typed EPR
        example.counter.context.stubs.types.CounterContextReference ref = new
example.counter.context.stubs.types.CounterContextReference();
        ref.setEndpointReference(epr);

        return ref;

```

```
}
```

Setting the Termination Time on a Resource

When the lifetime resource framework option is set on a service the resources of that service will be able to have tier lifetime managed. From the client side a

`SetTerminationTime()` or `Destroy()` grid call can be made to the service either setting the time to terminate or destroying the addressed resource. You can also call the set termination time directly on the resource inside the service. An example below, utilizing the same code as above will enable us create counters for users that will only be alive for 5 minutes:

```
public example.counter.context.stubs.types.CounterContextReference
createCounter() throws RemoteException {
    org.apache.axis.message.addressing.EndpointReferenceType epr = new
org.apache.axis.message.addressing.EndpointReferenceType();
    example.counter.context.service.globus.resource.BaseResourceHome home =
null;
    org.globus.wsrf.ResourceKey resourceKey = null;
    org.apache.axis.MessageContext ctx =
org.apache.axis.MessageContext.getCurrentContext();
    String servicePath = ctx.getTargetService();
    String homeName = org.globus.wsrf.Constants.JNDI_SERVICES_BASE_NAME +
servicePath + "/" + "counterContextHome";

    try {
        javax.naming.Context initialContext = new
javax.naming.InitialContext();
        home =
(example.counter.context.service.globus.resource.BaseResourceHome)
initialContext.lookup(homeName);
        resourceKey = home.createResource();

        // Grab the newly created resource

        example.counter.context.service.globus.resource.CounterContextResource
thisResource =
(example.counter.context.service.globus.resource.CounterContextResource)home.fi-
nd(resourceKey);

        // This is where the creator of this resource type can set whatever
needs
        // to be set on the resource so that it can function appropriately for
instance
        // if you want the resource to only have the query string then there
is where you would
        // give it the query string.

        // sample of setting creator only security. This will only allow the
caller that created
        // this resource to be able to use it.

        //thisResource.setSecurityDescriptor(gov.nih.nci.cagrid.introduce.servicetoo-
ls.security.SecurityUtils.createCreatorOnlyResourceSecurityDescriptor());

        //set the termination time of this resource
        Calendar cal = new GregorianCalendar();
        cal.add(Calendar.MINUTE,5);
        thisResource.setTerminationTime(cal);
```

```

        String transportURL = (String)
ctx.getProperty(org.apache.axis.MessageContext.TRANS_URL);
        transportURL = transportURL.substring(0, transportURL.lastIndexOf('/'))
+1 );
        transportURL += "CounterContext";
        epr =
org.globus.wsrf.utils.AddressingUtils.createEndpointReference(transportURL, reso
urceKey);
    } catch (Exception e) {
        throw new RemoteException("Error looking up CounterContext home:" + e.getMessage(), e);
    }

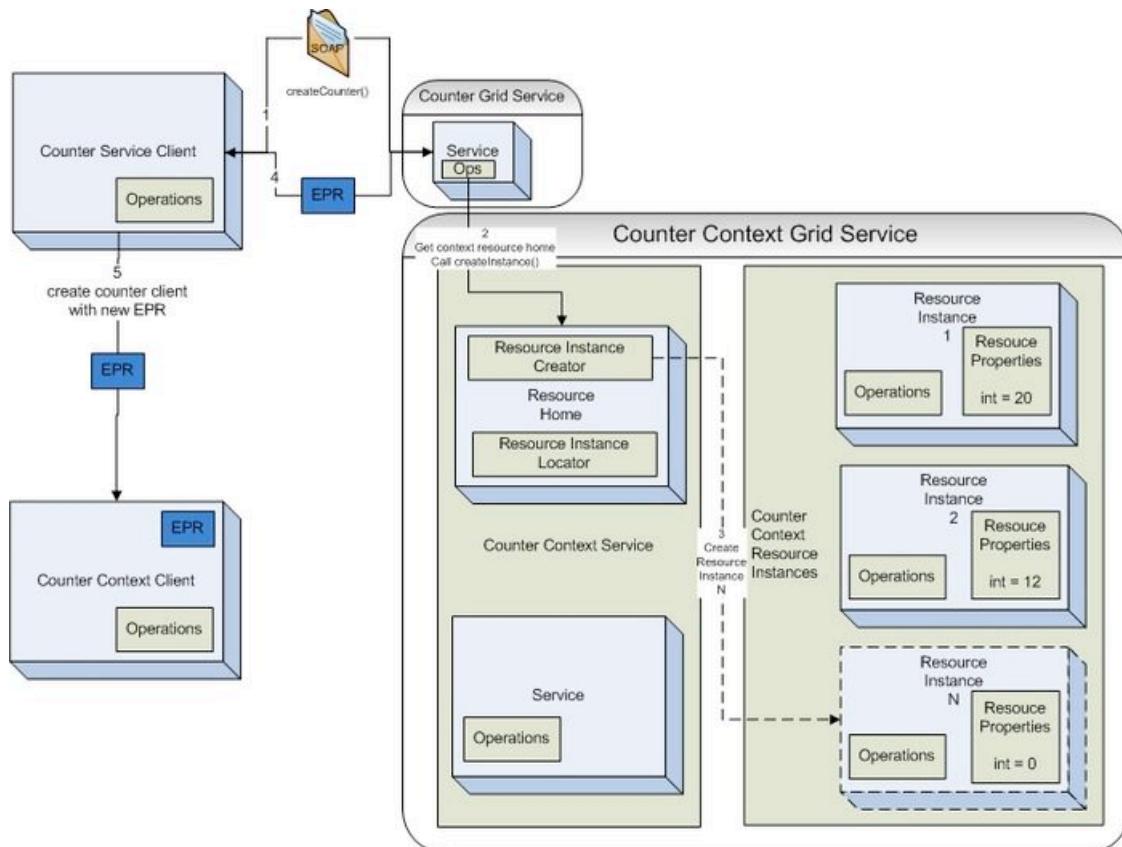
    //return the typed EPR
example.counter.context.stubs.types.CounterContextReference ref = new
example.counter.context.stubs.types.CounterContextReference();
ref.setEndpointReference(epr);

    return ref;
}

```

Utilizing the Factory Pattern

Click [here](#) to download the example service source code described below.

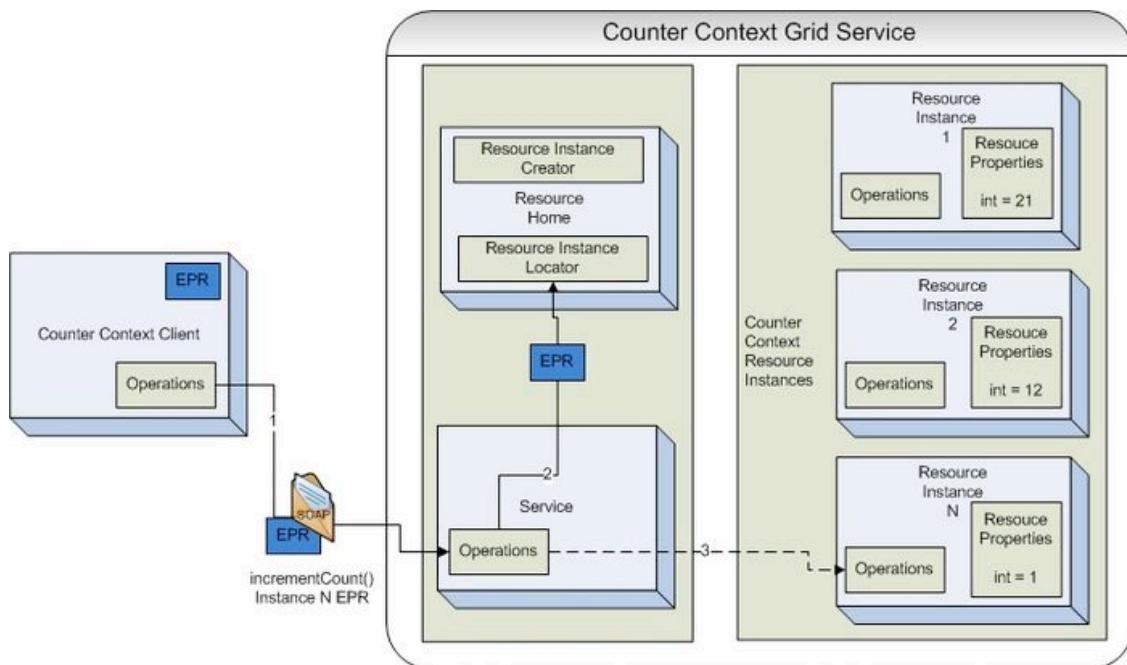


Utilizing Grid Service Factory Pattern

In order to create instances of a resource something must tell the resource home of the service to create them. This is typically done using the factory pattern. A service or

method is responsible for telling the resource to create a new instance of a resource type for the service. An example of the factory pattern in action is shown in the image above. A factory service, the Counter Grid Service, contains a create operation which is exposed as a grid service method. When the client invokes this method the Counter Grid Service will locate the resource home of the Counter Context Grid Service and ask it to create a new resource instance. Once this resource instance is created the resource home will return a pointer or address to this resource instance called an EndPointReference (EPR). This EPR is then returned from the create method back to the client so that it has a pointer to this new resource that it can act on.

In order to act on this resource it will need to construct a client which can talk to the grid service which represents this resource type. The Counter Context Client will be constructed using the EPR to address the specific resource context for the service. Next the user would like to call the `incrementCounter()` operation on the Counter Context Grid Service. The client will make the call to the grid service and the service will then look at the EPR sent over by the client. The service will then take the EPR and give it to the resource home and ask for the resource instance back which is represented by the particular EPR. The service will run its logic to increment the counter, which in this case is represented as a resource property of the resource.



Invoking a Statefull Grid Services

Below shows an example of the code that would be required to implement the `incrementCounter()` method in the `CounterContextServiceImpl.java`.

```
package example.counter.context.service;
import java.rmi.RemoteException;
/**
 * TODO:I am the service side implementation class. IMPLEMENT AND DOCUMENT ME
 *
 * @created by Introduce Toolkit version 1.1
 */
public class CounterContextImpl extends CounterContextImplBase {
```

```

public CounterContextImpl() throws RemoteException {
    super();
}

public void incrementCounter() throws RemoteException {
    int currentCounter;
    try {
        currentCounter = getResourceHome().getAddressedResource().getIntValue();

        currentCounter++;
        getResourceHome().getAddressedResource().setIntValue(currentCounter);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}

```

In Introduce 1.1, the `incrementCounter()` operation shown above is implemented in the service context implementation class (`CounterContextImpl`). The context operations implemented in this class are called each time a context client calls an operation. If the service context you are writing (as in the counter case above) maintains state for each context instance between calls, then that state can be accessed via the `getResourceHome().getAddressedResource()` call shown above. This call returns the specific instance of a `CounterContextResource` class that holds the state for the context client. In summary, the `CounterContextImpl` class is a stateless class that simply implements the context operations. The `CounterContextImpl` class uses the `CounterContextResource` instance specific to the EPR used by the client to access and maintain state for the client who is currently calling the context operation. In the above example, the state in the `CounterContextResource` is the current count retrieved in the impl via the `getIntValue` method (which is a service context property added via Introduce GDE).

Introduce Toolkit Developers

Creating an Update Site

Providing an update site is one way to enable Introduce users to be able to obtain your extensions or modified versions of Introduce. An update site is simply an address which points to a web folder that contains a `software.xml` file adhering to the schema. The `software.xml` file describes the packages available at the particular update site and the release zip which contains those packages.

```

<ns1:Software
  xmlns:ns1="gme://gov.nih.nci.cagrid.introduce/1/Software">
  <ns1:Introduce version="1.0"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/introduce_1-0.zip"
/>
  <ns1:Introduce version="1.1"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/introduce_1-1.zip"
/>
  <ns1:Extension version="1.1" compatibleIntroduceVersions="1.1"
displayName="caGrid Data Service Extension" name="data"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/data.zip"/>

```

```

<ns1:Extension version="1.1" compatibleIntroduceVersions="1.1"
displayName="caGrid Bulk Data Transfer" name="bdt"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/bdt.zip"/>
  <ns1:Extension version="1.1" compatibleIntroduceVersions="1.0,1.1"
displayName="caGrid Extensions Package" name="cabigextensions"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/cabigextensions.zip
"/>
    <ns1:Extension version="1.1" compatibleIntroduceVersions="1.0,1.1"
displayName="caGrid Cancer Data Standards Repository" name="cadsr"
zipFileURL="http://bmi.osu.edu/~hastings/introduce/software/cadsr.zip"/>
</ns1:Software>
```

Writing an Extension

When writing an extension it is first important to chose which type of extension that you want to write. A description of the extension framework and the important extension types is documented [here](#). There are extensions that add functionality to introduce and there are extensions designed to add functionality to a service. The most common introduce extension type is the service extension. This type of extension is designed to add functionality to a service. The best place to learn about extension is just to look at one. Here are a list of the service extension that can be found in the cagrid repository that are great places to start.

- cagrid/projects/data
- cagrid/projects/cabigextensions
- cagrid/projects/bdt
- cagrid/projects/transfer

Every extension revolves around its `extension.xml`. The `extension.xml` describes what type the extension is, and what are the classes that need to be invoked in order to execute the extension.

Bundling an Extension

In order to bundle up an extension so that it can be deployed to introduce the extension directory structure must be laid out in the following way:

- <top of zip file>
 - lib (any jars required to "execute" your extension)
 - <your extension name (not display name)>
- `extension.xml` (the `extension.xml` describing your extension)

Once this layout is complete the next step is to create the extension zip file. In order to do this all that is required is to create a zip file with no top level directory and the lib dir and <your extension name> dir as the only directories in the zip. So for instance, if the zip file were extracted there would be the two directories with no top level directory. Now that you have bundled up your extension so that it can be delivered and used by Introduce.

Chapter 6 caGrid Data Services

This chapter describes the caGrid Data Services infrastructure.

Topics in this chapter include:

- Overview on this page
- CQL on page 133
- Creating a Query on page
- Service Styles Architecture on page 146
- Federated Query Processor Usage Overview on page 149
- API Details on page 154

Overview

caGrid Data Services provide an object view of a data resource across the grid. The data resource is exposed through a well defined query method, which also relies on well defined query language objects to perform queries and return results as a strongly typed set. caGrid Data Services are designed to expose objects whose XML schemas are registered in the GME, and also expose metadata about those data objects derived from the caDSR.

Data Services also provide support for integration with alternate results delivery mechanisms such as WS-Enumeration. Enabling these features adds new query methods to the Grid-facing API.

CQL

CQL (Common/caGrid Query Language) is the caGrid query language used for all data services, and is defined in an XML document conforming to a well defined schema with the URI <http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery>.

The CQL Schema consists of the following main components (Figure 6-1):

- CQL Query
 - A simple wrapper element at the head of every CQL query document. It contains the target element.
- Target
 - The Target element is of the type Object, and describes the data type which the query will return.
- QueryModifier
 - An optional element modifying the returned result set. This modifier has a required attribute ‘countOnly’ which can tell the data service to return the number of results the query would return. The modifier optionally allows for a choice of a list of Attribute Names or a single Distinct Attribute to return. When the list of attribute names is specified, sets of tuples are returned containing the attribute names and corresponding values for each object

instance returned by the query. When distinct attribute is used, only unique attribute values are placed in the returned attribute sets.

- Object
 - The Object element contains the required attribute ‘name.’ This attribute’s value defines the caDSR class of the object. When the Object is the top level target of a CQL query, it identifies the data type that will be returned by the caGrid Data Service. The Object allows for a choice between three child elements. The possible child elements are Attribute, Association, and Group. Objects may have at most one of these child elements.
- Attribute
 - An Attribute type in CQL describes a restriction for an attribute of an Object. The Attribute contains three XML attributes, which define the restriction. The attribute ‘name’ defines the name of the attribute to be restricted. The attribute ‘value’ defines the restriction on the attribute. The attribute ‘predicate’ describes what type of restriction the Attribute defines. Allowable predicates are defined by the schema’s simple type ‘Predicate’, which defines an enumeration of allowable values. The predicate values are generally self-descriptive: “EQUAL_TO”, “NOT_EQUAL_TO”, “LIKE”, “LESS_THAN”, “LESS_EQUAL_TO”, “GREATER_THAN”, and “GREATER_EQUAL_TO.” Two additional predicates, “IS_NOT_NULL”, and “IS_NULL” check only for the presence or absence, respectively, of an attribute, and do not restrict its value at all. Therefore, any ‘value’ attribute will be ignored when using these predicates.
- Association
 - An association describes a related Object, which defines the associated Object’s restrictions for the query, as well as the relationship from one object to another. Specifically, it defines the relationship down the object model tree. The Association complex type is an extension of Object. The Association has a single, optional attribute named ‘roleName.’ This attribute identifies which associated object field the Association is defining. For example, a person may have more than one address, perhaps business and home. To perform a restriction against the home address, a query must specify the home address role name for the associated object. If the query omits the role name, such a query becomes ambiguous, as there is more than one field of Person which has a type of Address. In this case, the data service will throw a MalformedQueryException explaining that the requested association is ambiguous. In the case of an object where there is only one field of a given type, the roleName attribute may be omitted, and the data service will resolve the correct name as the query is processed.
- Group
 - Groups define logical joints of two or more conditions, and operate against the Object to which they are attached. Groups must have two or more children, which may be a mixture of type Attribute, Association, or Group. Groups also have an attribute named ‘logicOperator,’ whose type is defined in the schema’s simple type LogicalOperator. This type is an enumeration of the values “AND” and “OR.” The operator is applied to all children in the group. The “AND” operator requires that all conditions in the group be true for

the group to evaluate as true. The “OR” operator requires that any condition in the group evaluate as true.

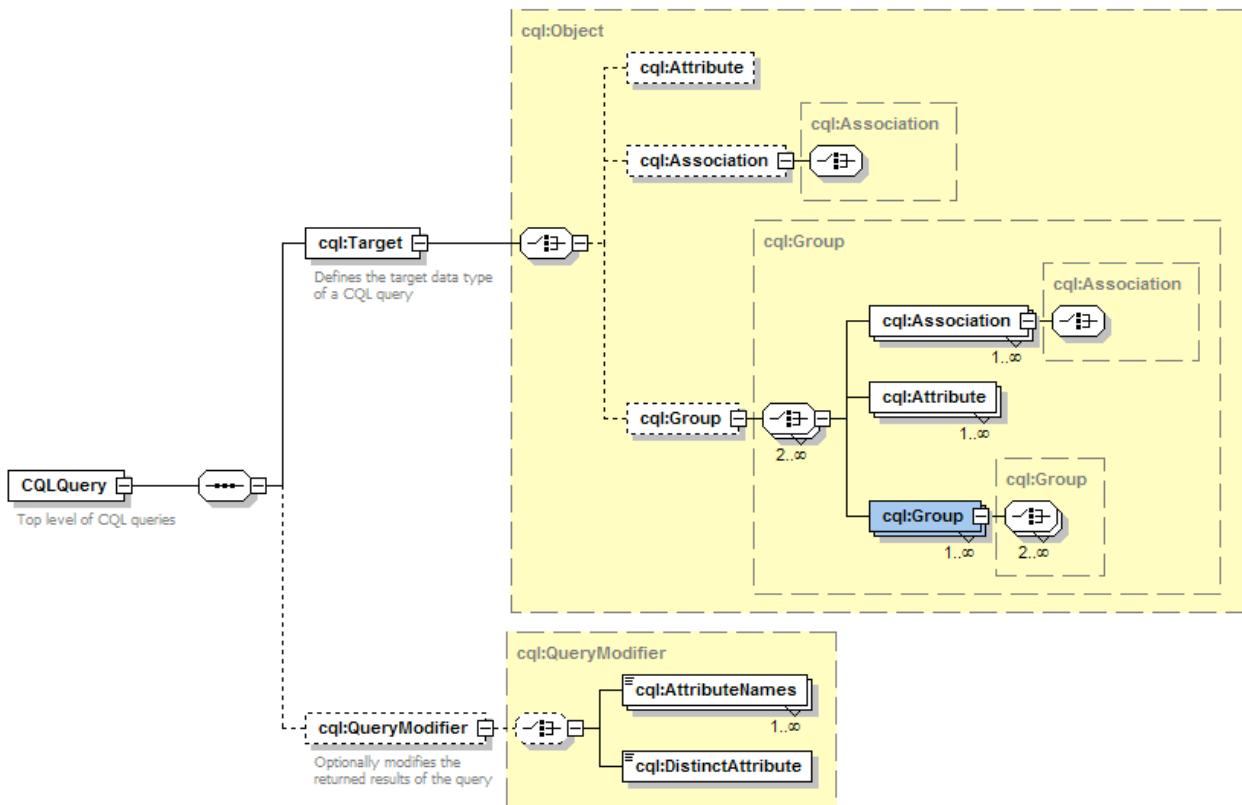


Figure 6-1 CQL Schema

Several example CQL queries can be found on the caGrid wiki at:
http://www.cagrid.org/mwiki/index.php?title=Data_Services:CQL

Creating a Query

Data services in caGrid use CQL to compose queries. A query can be produced programmatically, building up parts of the query using the supplied object model:

Programmatic query building

```

gov.nih.nci.cagrid.cqlquery.CQLQuery query =
    new gov.nih.nci.cagrid.cqlquery.CQLQuery();
gov.nih.nci.cagrid.cqlquery.Object target =
    new gov.nih.nci.cagrid.cqlquery.Object();
target.setName(gov.nih.nci.cabio.domain.Gene.class.getName());
gov.nih.nci.cagrid.cqlquery.Attribute symbolAttribute =
    new gov.nih.nci.cagrid.cqlquery.Attribute(
        "symbol",
        gov.nih.nci.cagrid.cqlquery.Predicate.LIKE,
        "%IL%");
target.setAttribute(symbolAttribute);
query.setTarget(target);
  
```

Loading queries from files

Alternatively, a CQL query can be loaded from a string of XML text or an XML file on disk and deserialized into the object model:

```
// from a string
CQLQuery query2 = (CQLQuery)
    gov.nih.nci.cagrid.common.Utils.deserializeObject(
    new StringReader("<CQLQuery ... />"), CQLQuery.class);
// from a file
CQLQuery query3 = (CQLQuery)
    gov.nih.nci.cagrid.common.Utils.deserializeObject(
    new FileReader(cqlFile), CQLQuery.class);
```

Generic Data Service Clients

The caGrid Data Services infrastructure supplies three basic client classes which can be used to invoke any arbitrary caGrid Data Service. This capability is due to the query methods of all data services being defined in a common, well known WSDL which each unique service instance imports.

The basic data service client, which is capable of invoking any caGrid Data Service, is the class `gov.nih.nci.cagrid.data.client.DataServiceClient`. The class defines the `query()` method, which takes a CQL Query as its single parameter and returns a CQL Query Results instance. A sample usage of this class is provided below:

```
import gov.nih.nci.cagrid.common.Utils;
import gov.nih.nci.cagrid.cqlquery.CQLQuery;
import gov.nih.nci.cagrid.cqlquery.Object;
import gov.nih.nci.cagrid.cqlresultset.CQLQueryResults;
import gov.nih.nci.cagrid.data.DataServiceConstants;

public class SampleDataServiceInvocation {

    public static void main(String[] args) {
        try {
            DataServiceClient client = new DataServiceClient(args[0]);
            CQLQuery query = new CQLQuery();
            Object target = new Object();
            target.setName("some.class.name");
            query.setTarget(target);
            CQLQueryResults results = client.query(query);
            Utils.serializeDocument("myResults.xml", results,
                DataServiceConstants.CQL_RESULT_COLLECTION_QNAME);
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
```

This small sample will create a new data service client using a URL specified on the command line and submit a query to it for all objects of the type `some.class.name`. The results will be stored on disk in an XML document named `myResults.xml`. The `DataServiceConstants` class used in this example provides static Strings and QNames used throughout the data service infrastructure. The constant `CQL_RESULT_COLLECTION_QNAME` is the QName which defines the XML type for result sets returned from the data service's query method.

Additionally, the caGrid Data Services infrastructure provides clients that can connect to data services which support WS-Enumeration and the caGrid Bulk Data Transfer infrastructure. Respectively, these clients are

`gov.nih.nci.cagrid.data.enumeration.client.EnumerationDataServiceClient` and `gov.nih.nci.cagrid.data.bdt.client.BDTDataServiceClient`. These clients provide public APIs which return an `EnumerationContext` instance or a `BulkDataHandlerReference` respectively. These return types may be used to start up an instance of the Globus provided `ClientEnumIter` class, or make use of the caGrid Bulk Data Transfer Client directly.

The client classes provided with the data service infrastructure, as well as any other clients generated by the Introduce toolkit, should not be assumed to be thread safe. Each thread communicating with a data service should have its own instance of the client class. Since client instances are unique, multiple data service clients may be used within the same thread or JVM to communicate with multiple data services simultaneously.

Client Side Utilities

The caGrid Data Services infrastructure provides a number of utility classes to make invocation of remote data services a simpler process for the application developer. The package `gov.nih.nci.cagrid.data.utilities` contains utilities can invoke standard, enumeration, and BDT data services, as well as tools for handling domain models and working with wsdd and castor mapping files.

Iterating Query Results

When a query is performed using the standard caGrid Data Service client's query method, a `CQLQueryResults` object is returned. This object is a container for both the results themselves and some information pertaining to their type. This container can contain object results, attribute name/value pairs, or a count of the total number of items in the result set. The difficulty of manipulating a container which may contain such a wide variety of result types stored in it may be handled by an iterator class provided with the data service infrastructure.

The interface `DataServiceIterator` specifies a single `query()` method, which takes a CQL Query and returns an instance of `java.util.Iterator`. The iterator can be used to walk through the results of a query issued to a data service. Three concrete implementations of the `DataServiceIterator` interface are provided, each for communicating with a different type of data service. The `DataServiceHandle` class can be used to invoke a standard caGrid Data Service, while the `EnumerationDataServiceHandle` and `BdtDataServiceHandle` classes are designed for WS-Enumeration and Bulk Data Transfer supporting data services, respectively.

Additionally, this package contains an Iterator utility for handling `CQLQueryResults` instances directly. The class `CQLQueryResultsIterator` implements the `java.util.Iterator` interface, and has three constructors. The choice of constructor affects the behavior of calling the `next()` method.

- `CQLQueryResultsIterator(CQLQueryResults)`
 - Creates an Iterator over the results which will return materialized objects deserialized using the default type mappings.

- CQLQueryResultsIterator(CQLQueryResults, boolean)
 - Creates an Iterator over the results, and the value of the Boolean parameter indicates if XML strings should be returned from the *next()* method.
 - If the Boolean value is true, XML text of each item is returned, otherwise the results will be deserialized using the default type mappings.
- CQLQueryResultsIterator(CQLQueryResults, InputStream)
 - Creates an Iterator over the results, and expects the InputStream will point to a client or server side wsdd.
 - The contents of this wsdd file will be used to configure deserialization of the objects contained in the results.

The class `gov.nih.nci.cagrid.data.utilities.CQLQueryResultsIterator` implements the `java.util.Iterator` interface, and so can be used in a `while()` loop like any other iterator over a Java collection. Depending on what the query to the data service asked for, calls to the `next()` method of this iterator will return different types of objects.

- If the query was for object results, then:
 - The iterator returns objects of the type specified as the target for the query.
 - Objects which require custom serialization and/or deserialization require that the iterator be configured with an InputStream to the client-config.wsdd file containing the type mappings for the objects.
 - Alternatively, the iterator can be configured to return only the XML representation of those objects.
- If the query was for attribute results, including distinct attributes, then:
 - Each successive call to `next()` returns an **array** of `TargetAttribute` types.
 - These types contain the name of an attribute and its value.
 - The value in the `TargetAttribute` instance will be null if the value was null on the object satisfying the query. Each array of `TargetAttributes` corresponds to one object instance which satisfied the CQL query criteria.
- If the query was for a count of object instances, then:
 - The iterator returns a single `java.lang.Long` value.

An example usage of this iterator is below:

```
import gov.nih.nci.cagrid.cqlquery.CQLQuery;
import gov.nih.nci.cagrid.cqlresultset.CQLQueryResults;
import gov.nih.nci.cagrid.data.client.DataServiceClient;
import gov.nih.nci.cagrid.data.utilities.CQLQueryResultsIterator;

import java.util.Iterator;

public class SampleDataServiceInvocation {

    public static void main(String[] args) {
        try {
            DataServiceClient client = new DataServiceClient(args[0]);
            CQLQuery query = new CQLQuery();
            // build up the query
            CQLQueryResults results = client.query(query);
```

```

Iterator iter = new CQLQueryResultsIterator(results,
    SampleDataServiceInvocation.class.getResourceAsStream(
        "client-config.wsdd"));
while (iter.hasNext()) {
    java.lang.Object result = iter.next();
    // do something with the result object
}
} catch (Exception ex) {
    ex.printStackTrace();
    System.exit(1);
}
}
}

```

More information on the [caGrid Data Services client APIs](#) may be found on this wiki.

Most Current Information and Examples

The most current information regarding the caGrid Data Services client side APIs and utilities may be found on the caGrid.org wiki page:

(http://www.cagrid.org/mwiki/index.php?title=Data_Services:Client_API)

Utility Classes

Utilities

The caGrid data services infrastructure includes several utility classes which can be used to ease development and use of data services. These classes are found in the `gov.nih.nci.cagrid.data.utilities` package distributed with the data service infrastructure.

CQLResultsCreationUtil

This class provides convenience methods for creating CQLQueryResults instances for object results, attribute results, and a counting result. A convenience method for identifier results may be added in the future. The class provides three public static methods, one for each type of results currently supported.

- `public static CQLQueryResults createObjectResults(List objects, String targetName, Mappings classToQname)`
 - `objects` – a list of Java objects to be placed in a new CQLQueryResults object.
 - `targetName` - the name of the class targeted by the query which produced these object results. All items in the `objects` list should be of this type.
 - `classToQname` – a mapping from class name to QName. This is a generated Java bean from the XML schema for the data service infrastructure and contains an array of name/value pairs that map class names to QNames.
- `public static CQLQueryResults createAttributeResults(List attribArrays, String targetClassname, String[] attribNames)`

- attribArrays - a List of Object arrays. Each array should have one value for one attribute of an object. These values may be null. The values must be in an order corresponding the ordering of attribute names
- targetClassname - the name of the class targeted by the query which produced these attribute results. All attribute arrays should have some from this type.
- attribNames - the names of the attributes returned by the query. These should be in the same ordering used by the attribute arrays.
- public static CQLQueryResults createCountResults(long count, String targetClassname)
 - count - the number of resulting items (objects, attribute sets) from a query
 - targetClassname - the name of the class which was the target of the query

DataServiceIterator

The data service iterator is an interface which provides for a query to be submitted to a data service and an Iterator over the result set to be returned. There are two implementations of this interface; one for the standard data service and one for data services with enumeration enabled.

- DataServiceHandle
 - The data service handle is the implementation of the data service iterator class for a base caGrid Data Service. It has three constructors, all of which take a DataServiceClient instance. The default constructor needs only this parameter. The other two constructors should be used when custom serialization and deserialization of types has been specified for the service. The extra parameter can be either the filename of a wsdd file containing this mapping information, or an InputStream to the same information.
- EnumDataServiceHandle
 - The enum data service handle is the implementation of the data service iterator interface for a WS-Enumeration enabled caGrid Data Service. It has two constructors, both of which take an enumeration data service client instance. The default constructor needs only this parameter. The second constructor takes an IterationConstraints instance, which contains information about how data should be requested from the enumeration data service.
- BdtDataServiceHandle
 - The BDT data service handle is an implementation of the data service iterator interface to be used with a BDT-enabled caGrid Data Service. Its behavior is the same as that of the enum data service handle, except that it handles the additional invocation of the BDT context to support enumeration internally.

DomainModelUtils

The domain model utils provide a means to extract useful information from a domain model.

- public static UMLClass getReferencedUMLClass(DomainModel model, UMLClassReference reference)

- To save on document size, domain models do not duplicate class information when an association is defined, but rather use class references based on ID values. These reference values can be traced back to their original UML Class instance with this function.
- public static UMLClass[] getAllSuperclasses(DomainModel model, String className)
 - Superclasses of a UML Class can be determined by traversing UML class references and generalization information. There are two methods which perform this task in the Domain Model Utils class. One uses a class name and the other extracts the name from an UMLClass instance and passes it to the other.

WsddUtil

The wsdd utility class contains functions to set parameters on a wsdd file. This class is used internally to the Introduce data service extension to edit the wsdd files and change the castor mapping file name.

- public static void setGlobalClientParameter(String clientWsddFile, String key, String value)
 - clientWsddFile - the name of the client side wsdd file to edit. When edits are complete, the changed file is saved to the same location.
 - key - the key of the parameter. This is the name by which the parameter can be accessed.
 - value - the value stored in the parameter
- public static void setServiceParameter(String serverWsddFile, String serviceName, String key, String value)
 - serverWsddFile - the name of the server side wsdd file to edit. When edits are complete, the changed file will be saved to the same location
 - key - the key of the parameter. This is the name by which the parameter can be accessed
 - value - the value stored in the parameter

Validation Tools

The caGrid Data Services infrastructure provides for validation of queries with respect to the domain model exposed by a service and the CQL schema, as well as query results for validity with respect to the exposed data types.

CQL Query Syntax

The caGrid Data Service infrastructure provides mechanisms to validate CQL queries for syntactic correctness. While the Axis engine prevents malformed XML from ever being turned into CQL objects, it does not handle XML that does not conform to certain schema restrictions. For example, Axis does not prevent populating multiple child elements of an XML schema 'choice'. For this reason, CQL syntax validation can be enabled on a caGrid data service. This mechanism will reject invalid queries before they ever reach a CQL Query Processor implementation, saving the processor's developer

from having to handle them. This same validation can be performed either on the client side or offline completely by using the query validation utilities. For syntax validation, the interface `gov.nih.nci.cagrid.data.cql.validation.CqlStructureValidator` is provided, as are two implementations of this interface. The interface provides the `validateCqlStructure()` method, which takes a single `CQLQuery` instance parameter, and throws a `MalformedQueryException` if an error is encountered. The default implementation of this interface is the `gov.nih.nci.cagrid.data.cql.validation.ObjectWalkingCQLValidator` class. As its name suggests, this class walks through the CQL object model, seeking out inconsistencies with the published CQL schema. This class also has a `main()` method, which allows it to be run from the command line with a list of CQL query XML files specified as arguments. The data service infrastructure uses this class by default when query validation is enabled. This can be changed to any other class which implements the `CqlStructureValidator` interface by editing the value of the `dataService_cqlValidatorClass` service property in a generated data service.

Domain Model Conformance

The Data Service infrastructure also provides mechanisms to validate a structurally sound CQL query against a Domain Model to ensure its restrictions are supported by the domain model's exposed structure. Domain Model validation may be enabled for a caGrid data service, and will be performed on every query submitted to the service before it is passed to the CQL query processor. The interface

`gov.nih.nci.cagrid.data.cql.validation.CqlDomainValidator` is provided, along with a single implementation. The interface provides the `validateDomainModel()` method, which takes a single `CQLQuery` instance parameter, and throws a `MalformedQueryException` if an error is encountered. The lone implementation provided with the caGrid Data Service infrastructure is the `gov.nih.nci.cagrid.data.cql.validation.DomainModelValidator` class. Like the CQL validation instance, this class has a `main()` method, which allows it to be run from a command line. The arguments should be first a domain model XML file, then a list of CQL query files to be validated. The data service infrastructure uses this class when domain model validation is enabled. This implementation may be substituted for another by editing the value of the `dataService_domainModelValidatorClass` service property in a generated data service.

Results Validation

The data service infrastructure also provides a means to both validate the results of a CQL query against a known set of targets, and to determine what target data types are allowed to be returned by a caGrid Data Service. Every data service exposes a schema through its WSDL that enumerates the data types which may be returned by the data service. This schema appears in generated services under the `schemas/<ServiceName>` directory as `<ServiceName>_CQLResultTypes.xsd`.

The utility class

`gov.nih.nci.cagrid.data.utilities.validation.CQLQueryResultsValidator` has been provided to both retrieve this file and verify that a `CQLQueryResults` instance conforms to this schema. An instance of this class can be constructed with either the full path to a data service's WSDL file, or an endpoint reference to a running data service.

The validator exposes two public methods:

This method locates the restriction XSD file and saves its contents into the file specified.

- ```
public void saveRestrictedCQLResultSetXSD(File fileLocation) throws SchemaValidationException
```

  - `fileLocation` - a file into which the restriction XSD will be saved.
- ```
public void validateCQLResultSet(CQLQueryResults resultSet) throws SchemaValidationException
```

 - `resultSet` - a set of results generated by a query into a caGrid Data Service. The object contents of this result set will be processed against the restriction XSD.

The CQLQueryResultsValidator class also has a `main()` method, which takes two arguments. The first argument is a URL to a caGrid Data Service, which will be used to retrieve the result restriction schema. The second argument should be the filename of a CQLQueryResults instance serialized to an XML document.

CQL Query Processors

Overview

The CQL Query Processor is a pluggable implementation which handles the details of processing CQL against some backend data source and produces a CQLQueryResults instance. The particular implementation used is determined by a value in the service's deployment properties, and an instance of the processor is loaded at runtime via reflection. The query processor may optionally supply a set of properties via the `getRequiredParameters()`. These properties may be configured prior to deployment of the service, and are passed into the query processor when it is first instantiated via the `initialize()` method. Additionally, the query processor may implement the method `getPropertiesFromEtc()`. This method returns a `java.collections.Set` containing a subset of keys from the `getRequiredParameters()` method whose values should be returned as file system paths relative to the etc directory of the deployed grid service.

When a query is issued to the data service, the query will be passed along to the CQL Query Processor instance's `processQuery()` method. This method may throw both a `QueryProcessingException` in the case of an error in handling the query and a `MalformedQueryException` in cases where the query was found to be invalid for any reason.

Implementation

All query processor implementations are required to extend the abstract base class `gov.nih.nci.cagrid.data.cql.CQLQueryProcessor`. This base class declares several methods which are meant to be overridden, however the only method a query processor is required to implement is the `processQuery()` method. This method takes a CQL query and returns a CQLQueryResults instance. This method is declared abstract in the base class, which enforces this implementation requirement. Generally, this method should be able to translate CQL into whatever native query language is required by the back end data resource, and translate the result set into a CQLQueryResults instance.

CQL Query Processors are designed to be configurable at runtime by a set of properties. These properties are modifiable via the data service extension to the Introduce toolkit, or manually by editing a configuration file once a service has been built.

The query processor base class `gov.nih.nci.cagrid.data.cql.CQLQueryProcessor` declares several methods which may be overridden to add functionality and configurability to a query processor:

- `getRequiredParameters()`
 - This method returns a `java.util.Properties` instance, containing keys and default values used in configuration of a query processor. A query processor implementation may, for example, require a database username and password, both of which could be defined here.
 - All property keys must be valid Java identifiers meaning that there cannot be any spaces or punctuation in the key.
 - When a query processor is selected in the Introduce Toolkit, these properties are displayed and the service developer is given the opportunity to edit them. During this process, the query processor class is instantiated and the `getRequiredParameters()` method is invoked. For this reason, the query processor's constructor and `getRequiredParameters()` method should **not** perform any operations other than returning this required properties set.
- `getPropertiesFromEtc()`
 - This method returns a `java.util.Set` containing a subset of keys from the `getRequiredParameters()` method whose values should be returned as file system paths relative to the etc directory of the deployed grid service.
- `getConfigurationUiClassname()`
 - This method returns a string identifying the class name of the query processor's configuration UI. This UI must extend from the base class `gov.nih.nci.cagrid.data.cql.ui.CQLQueryProcessorConfigUI`. This UI may be used to perform more sophisticated configuration of a query processor than can be provided with a simple set of property keys and values.
- `initialize()`
 - This method takes a `java.util.Properties` instance and a `java.io.InputStream`. The properties contain the configured version of the properties specified by the return value of `getRequiredParameters()`, and the input stream loads the `server-config.wsdd` file. This may be useful for serializing or deserializing object instances, and is required by some of the query results utilities.
 - If this method is overridden, be sure to call `super.initialize()` first, as this will both validate and store the properties and wsdd input streams.
 - This method is called once per query processor instance, and there is only one instance of the query processor in each data service.
 - Only after this method has been called will `getConfiguredParameters()` return value be non-null. The same rule applies to the `getConfiguredWsddStream()` method.

The query processor base class has two protected methods which provide access to any user configured parameters and an input stream to the server side wsdd configuration file. The method `getConfiguredParameters()` returns a `java.util.Properties` instance containing all the keys defined in the properties returned by `getRequiredParameters()`,

but with either the default or a developer configured value associated with each. The method `getConfiguredWsddStream()` returns an `InputStream` instance which will read in the contents of the server side wsdd configuration file. The call to the query processor's initialize method, and in turn the population of these values, occurs when the data service is first instantiated, typically when the container is started. Calls to these methods before this time will return null. For this reason, the constructor of the CQL Query Processor implementation must be fairly simple, and initialization of any resources required delayed until the `initialize()` method has been called.

```
/*
 * Processes the CQL Query
 * @param cqlQuery
 * @return The results of processing a CQL query
 * @throws MalformedQueryException
 *         Should be thrown when the query itself does not conform to the
 *         CQL standard or attempts to perform queries outside of
 *         the exposed domain model
 * @throws QueryProcessingException
 *         Thrown for all exceptions in query processing not related
 *         to the query being malformed
 */
public abstract CQLQueryResults processQuery(CQLQuery cqlQuery)
throws MalformedQueryException, QueryProcessingException;
```

The only method which is absolutely required to be implemented by CQL query processors is the `processQuery()` method. This is the method which executes the CQL query against its data source and generates an appropriate set of results. There are utilities (discussed earlier) to make generation of this result set a simpler process. At the time this method is called, the return values of `getConfiguredParameters()` and `getConfiguredWsddStream()` will be non-null.

The `processQuery()` method throws both a `MalformedQueryException` and a `QueryProcessingException`. Malformed query exceptions should be thrown under conditions where the query is somehow incorrect syntactically, or uses features of the CQL language which are not yet supported in the query processor implementation. If query syntax validation is enabled in the data service infrastructure, then it may be assumed that all queries reaching the `processQuery()` method are at least well formed CQL. Query processing exceptions should be thrown when some error occurs which prevents successful resolution of the query request. These conditions may include database errors, file system problems, or misconfiguration of properties.

The data service infrastructure provides a few utilities which may prove helpful when implementing a CQL Query Processor:

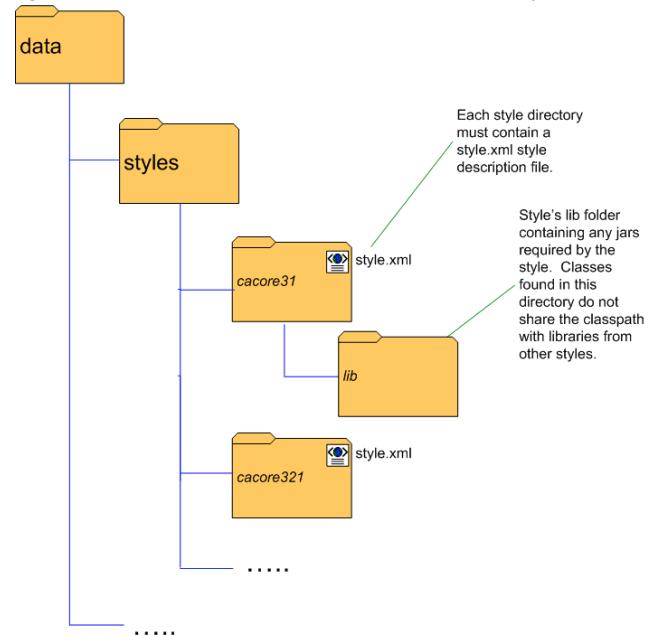
- `gov.nih.nci.cagrid.data.utilities.CQLResultsCreationUtil`
 - This class provides methods for creating object, attribute, and count results given a list of objects, attributes, or a result count. The return value from each of its static methods is a `CQLQueryResults` object, which may be returned by the `processQuery()` method without any further implementation required.
- `gov.nih.nci.cagrid.data.utilities.DomainModelUtils`
 - This class provides methods for navigating and extracting information from a Data Service's domain model, such as all superclasses of a given class. This information may be useful for navigating object hierarchies.

Service Styles Architecture

Data service styles may be added to the data service extension to provide additional functionality to the service creation and configuration processes, and are selected by the service developer when a service is first created. Styles may be installed at any time after the primary caGrid Data

Services extension has been installed by adding to the styles directory found in the installed data service extension directory. Each style must provide its own directory in which files it uses will be placed, but no restriction is made on the naming of these directories. At the top level of each style directory, a style.xml file must exist, describing the style. This document describes the style's name, which caGrid and Data Service versions it is compatible with, and information on which classes are to be loaded for each component of the style. If the service developer selects no style at service creation time, the service is created with only the standard data services

components and query method, and ready to have a custom domain model, query processor, and other data service requirements selected and configured.



Functionality Extended by Styles

Data Service styles may add functionality to any or all of the following areas of service development with the Introduce toolkit:

- Creation Wizard
 - The service style may supply a list of wizard panels to be displayed and chained together in a wizard-like fashion to break the setup process for the service style into a series of steps. These panels will be shown in a wizard dialog when a service style is selected at service creation time.
- Post-creation processing
 - Just as Introduce extensions may add functionality to the service creation process, data service styles may add processing capabilities to this step.
- Modification User Interface
 - The style may supply a graphical panel which will be added to the data service tab in the Introduce service modification viewer. This tab can be used to configure any style-specific options in the service.
- Post-code generation processing
 - The style may add functionality to the code generation process of service modification. This processing will be invoked each time the service is modified and saved in Introduce.

Developing Data Service Styles

The caGrid Data Service styles infrastructure requires style implementations to conform to a specific format, and requires that the classes which compose the style's implementation inherit from certain classes.

Style Description Document

A single XML document describes the style to the data services infrastructure. This document must be named `style.xml`, and placed in a **unique directory** under the `<introduce>/extensions/data/styles` directory. This style description document must conform to the schema provided with the data services infrastructure with the namespace <http://gov.nih.nci.cagrid.data/Style>, which can be found in the caGrid source repository.

The following is an example of the style description document:

```

<DataServiceStyle xmlns="http://gov.nih.nci.cagrid.data/Style" name="caCORE SDK
v 3.2(.1)" version="1.2">
    <CompatibleCaGridVersions>
        <Version>1.1</Version>
        <Version>1.2</Version>
    </CompatibleCaGridVersions>
    <StyleDescription>Configures the data service to utilize a data source
generated by the cacore SDK, versions 3.2 and 3.2.1</StyleDescription>
    <CreationWizardPanels>
        <CreationWizardPanel
classname="gov.nih.nci.cagrid.data.style.cacore32.wizard.SDK32InitializationPan
el"/>
        <CreationWizardPanel
classname="gov.nih.nci.cagrid.data.style.cacore32.wizard.SDKClientSelectionPan
el"/>
        <CreationWizardPanel
classname="gov.nih.nci.cagrid.data.style.cacore32.wizard.AppserviceConfigPanel"
/>
        <CreationWizardPanel
classname="gov.nih.nci.cagrid.data.style.sdkstyle.wizard.DomainModelPanel"/>
        <CreationWizardPanel
classname="gov.nih.nci.cagrid.data.style.sdkstyle.wizard.SchemaTypesPanel"/>
    </CreationWizardPanels>
    <CreationHelper
postCreationClassname="gov.nih.nci.cagrid.data.style.cacore32.helpers.SDK32Post
CreationProcessor"/>
    <CodegenHelpers
postCodegenClassname="gov.nih.nci.cagrid.data.style.cacore32.helpers.SDK32PostC
odegenHelper"/>
    <ModificationUiPanel
classname="gov.nih.nci.cagrid.data.style.cacore32.modification.CaCoreVersionPan
el"/>
</DataServiceStyle>
```

Each part of this document influences the behavior or appearance of the style as the data service infrastructure executes it.

Name and Version

The root element of the style document attributes which define the name of this style, along with the version of the style. The name is displayed to the service developer at service creation time, allowing selection of a style. The name and version uniquely identify a style to the data service extension framework at runtime.

Compatible Versions

The style document allows for defining which versions of the caGrid Introduce toolkit and data service extension the style is compatible with. The document may define multiple `<Version>` elements indicating compatibility with multiple versions.

Style Description

The document allows for a brief description of the style to accompany the style itself. This information is displayed at data service creation time when the user is given a chance to select a style to use with their data service.

Creation Wizard Panels

This optional element defines a sequence of panels which will be loaded as part of the pre code generation process of creating a new caGrid Data Service. The panels are displayed in the order in which they are defined here, and can be used to configure aspects of the service style, or customize the underlying data service extension.

Each wizard panel's Java class must extend from the base class `gov.nih.nci.cagrid.data.ui.wizard.AbstractWizardPanel`. This class is essentially a JPanel with accessor methods for the Introduce service model and the data service extension data. The class also provides access to wizard functionality, such as enabling and disabling navigation buttons, and indicating that the wizard has been completed. Additionally, the base class has three abstract methods which must be implemented:

- `update()`
 - This method is called by the wizard when the panel is to be displayed. This provides a callback to update the UI and reflect changes to the model which may impact the appearance of the wizard panel.
- `movingNext()`
 - ***This method is new for caGrid 1.2***
 - This method is called by the wizard itself when the user clicks Next or Finish button to move to the next wizard panel or finish the wizard. This method gives the panel a chance to synchronize its state with that of the service model (ie. copying schemas and libraries, storing service properties).
- `getPanelTitle() -> String`
 - This method should return the title you wish to be displayed by the wizard when the panel is visible on the screen.
- `getPanelShortName() -> String`
 - This method should return a very short name to appear in the navigation buttons of the wizard. It should generally be one or two words long.

Creation Process Helpers

The data service creation process can be extended by the service style. The post-creation process helper is defined by the `CreationHelper` element. This element has an attribute called `postCreationClassname`, which defines the class name of the post-creation processor class. This class must implement the interface `gov.nih.nci.cagrid.data.style.StyleCreationPostProcessor`.

Code Generation Process Helpers

The data service code generation process can be extended by the service style. The pre code generation process, defined by the `preCodegenClassname` attribute, is executed *after* the data service pre code generation options have been completed, but before any other extensions have run. The post code generation process, defined by the `postCodegenClassname` is executed similarly.

The pre-codegen class must implement the interface

`gov.nih.nci.cagrid.data.style.StyleCodegenPreProcessor`, and the post-codegen class must implement the interface

`gov.nih.nci.cagrid.data.style.StyleCodegenPostProcessor`.

Service Modification Panel

The style can optionally include a panel to be loaded into the data service modification UI. This panel can provide information on configuration options specific to the style (as in the case of the caCORE SDK styles), manage configuration, or present any other information the style developer may choose.

This class must extend from the base class

`gov.nih.nci.cagrid.data.ui.DataServiceModificationSubPanel`. This class is essentially a JPanel with additional methods for accessing the underlying Introduce service model and the data service specific extension data.

Style Library Directory

Each style must provide any jar files it depends on for Java classes and resources in a directory named `lib` within the style directory itself. At runtime, classes referenced from the style document are loaded from the jars provided here. The classes contained in these jars may safely depend on other caGrid and Globus libraries, as the runtime classpath of the Introduce toolkit is appended to the library directory's contents.

Federated Query Processor Usage Overview

The caGrid Federated Query Infrastructure provides a mechanism to perform basic distributed aggregations and joins of queries over multiple data services. As caGrid data services all use a uniform query language, CQL, the Federated Query Infrastructure can be used to express queries over any combination of caGrid data services. Federated queries are expressed with a query language, DCQL, which is an extension to CQL to express such concepts as joins, aggregations, and target services. The infrastructure is composed of a core engine and grid services which provide access to and management of the use of the core engine.

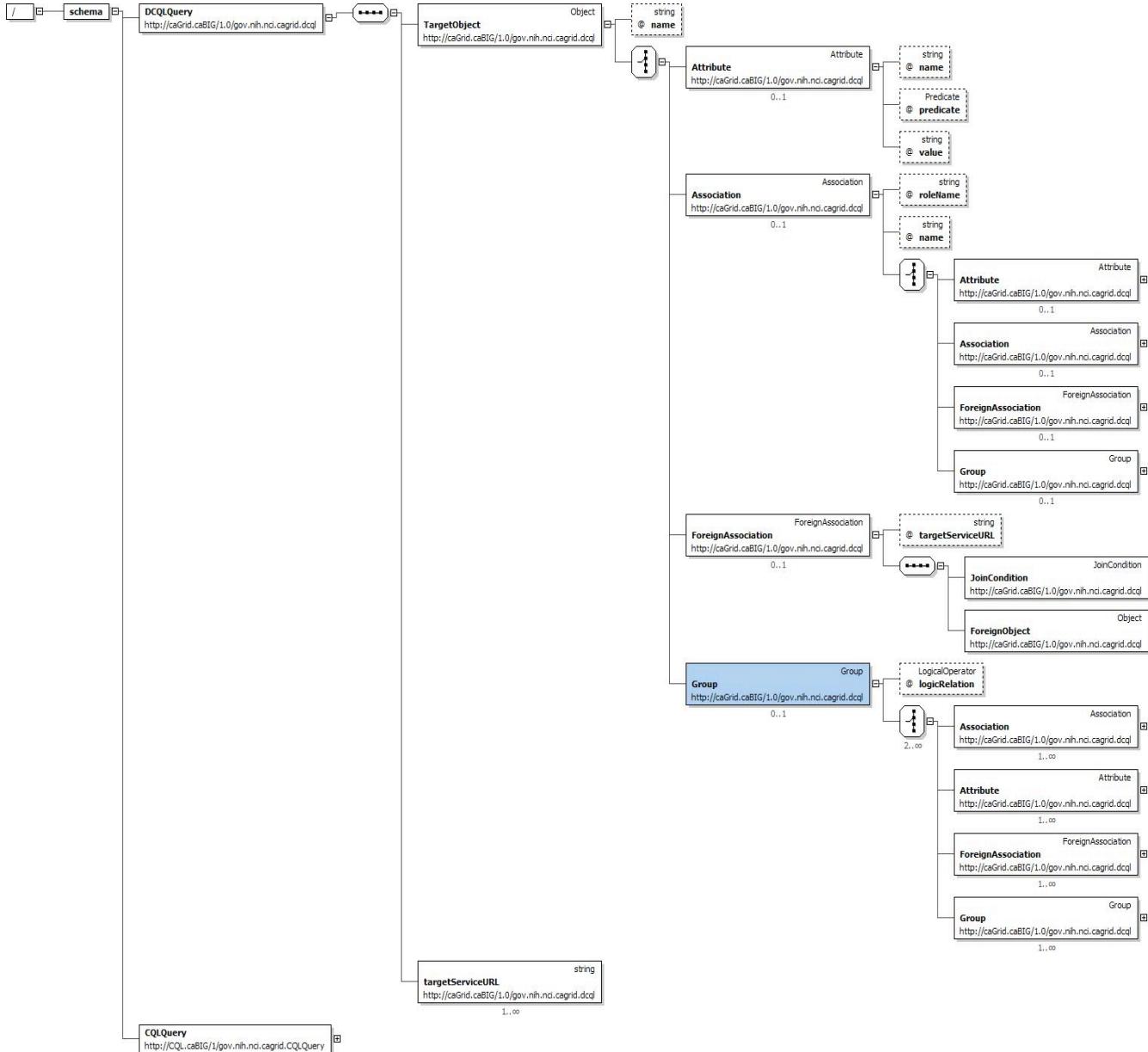


Figure 6-2 DCQL schema

DCQL, the language used to express federated queries, is an extension to CQL, the language used to express single data service queries. The DCQL schema is shown in Figure 6-2. Both CQL and DCQL use a declarative approach to describe the desired data by identifying the nature of the instance data with respect to its containing UML information model. That is, a query can be seen as identifying a class in a UML model, and restricting its instances to those which meet criteria defined over that class's UML attributes and UML associations.

The primary additions to CQL, which DCQL provides, are the introduction of the ability to specify multiple target services (aggregations), and the ability to specify object restrictions through relationships to objects on remote data services (joins). The other

primary difference between the languages is that CQL is context dependent, meaning the language must be interpreted against the service receiving the query, and DCQL itself specifies the context of the queries (by identifying the target services). As such, services accepting DCQL (such as the FQP service), generally do not expose any local data. Details on DCQL can be found in the Federated Query Processor design document.

An example DCQL query, represented in XML, is shown below in Figure 6-3. In this fictitious example, a PersonRegistry Data Service is joined with a StudyRegistry Data Service. The query specifies Persons in the PersonRegistry should be returned when they have a “ssn” that is equal to that of a Participant’s “patientSSN” and the Participant should have an “age” greater than 18. The specification of the target service can be seen on line 18 in the example (in this case only one service is targeted, though may could have been listed). Additionally, the “join” is specified starting on line 6, wherein the second target service is identified, and the join condition is defined. The join condition creates a link between the containing Object (in this case, Person), and an Object (in this case Participant, as defined on line 10) in the second target service. The condition specifies a predicate to be evaluated against an attribute in each of the two linked Objects (in this case Person.ssn and Participant.patientSSN). It is worth noting that as DCQL is a recursive language, the ForeignObject defined on line 10 could have also specified a join to a third Data Service, or other more complex criteria.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <DCQLQuery xmlns:cql="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery"
3   xmlns="http://caGrid.caBIG/1.0/gov.nih.nci.cagrid.dcql">
4   <TargetObject name="com.example.Person">
5     <Group logicRelation="AND">
6       <ForeignAssociation targetServiceURL=
7         "https://host2:8443/wsrf/services/cagrid/StudyRegistry">
8         <JoinCondition localAttributeName="ssn"
9           foreignAttributeName="patientSSN" predicate="EQUAL_TO"/>
10        <ForeignObject name="org.example.Participant">
11          <Attribute name="age" value="18" predicate="GREATER_THAN"/>
12        </ForeignObject>
13        <Attribute name="lastName" value="Foo%" predicate="LIKE"/>
14      </Group>
15    </TargetObject>
16    <targetServiceURL>
17      https://host1:8443/wsrf/services/cagrid/PersonRegistry
18    </targetServiceURL>
19  </DCQLQuery>
20

```

Figure 6-3 Example DCQL Query

The Federated Query Engine is a simple but powerful design. The main functionality of the engine is to process a DCQL query by converting it into regular CQL queries to the targeted data services, appropriately aggregating results. As such, all of the actual “joining” of data is offloaded to the remote data services. This allows the engine to be reused as a client API as no databases or complex service infrastructure is needed; it is simply a client-side querying tool. The engine requires no special support from data services. Each service which is contacted to satisfy the distributed query is only sent one or more standard, but potentially complex, CQL queries. It is possible to construct a DCQL query which is essentially a standard CQL query, with the addition of specifying one or more target data services. In this case, the engine simply “forwards” that query on

to the targeted services, and aggregates their results. Details on the implementation and query processing logic of the engine can be found in the Federated Query Processor design document.

The Federated Query Processing Infrastructure contains three main client-facing components: an API implementing the business logic of federated query support, a grid service providing remote access to that engine, and a grid service for managing status and results for queries that were invoked asynchronously using the query service.

Federated Query Engine

For clients not wishing to use the grid service, the `FederatedQueryEngine` is the client-facing entry point to the engine. It provides two methods which accept DCQL queries, and return the results. Each of the two methods provides a different variant on how results are represented. The first method is the `executeAndAggregateResults` method, which returns the standard `CQLQueryResults` (the same result type returned by data services' `query` method). Each `CQLResult` obtained from each targeted data service is merged into an aggregate list, and a master `CQLQueryResults` object is constructed which contains them all. The information about which result came from which data service is lost in this scenario, but this provides the ability to reuse existing data service tooling and APIs when that information is not relevant.

In cases where it is important to know from which data service a given result came, the second query method called `execute` can be used. This method returns a new type called `DCQLQueryResultsCollection`. The `DCQLQueryResultsCollection` contains a list of `DCQLResult`, wherein each `DCQLResult` specifies a `CQLQueryResults` object, and the data service URL from which it came. That is, the result type is a collection of tuples containing the standard data service results, and that service's URL.

Both query methods will throw a `RemoteDataServiceException` in the event a queried data service returns invalid results (such as the wrong target class type), or if a data service itself throws an exception when being queried, or if there is any problem querying the data service. Additionally, a `FederatedQueryProcessingException`, which is the parent class of `RemoteDataServiceException`, may be thrown if there is a problem processing the query itself.

Federated Query Processor Service

The Federated Query Processor service is the main service interface to the federated query engine. It provides three query execution operations. The first two are: `execute` which takes a DCQL query and returns a `DCQLQueryResultsCollection`, and `executeAndAggregateResults` which returns a `CQLQueryResults`. These are both simple grid service wrappers for the corresponding methods in the `FederatedQueryEngine` API. The third operation, `executeAsynchronously`, provides asynchronous, non-blocking, access to the `execute` method, and returns a `FederatedQueryResultsReference`. The `FederatedQueryResultsReference` is a typed container for an EPR to the Federated Query Results service. The Federated Query Results service client API can be used to subsequently retrieve the `DCQLQueryResultsCollection`.

Federated Query Results Service

The Federated Query Results service is the service responsible for providing access to query results and processing status for asynchronously executed queries. The service can only be contacted with a resource-qualified EPR, provided by the Federated Query Processor service. Whenever the query processor service is requested to execute an asynchronous query, a Resource is created and an EPR, which identifies that Resource in the results service, is returned. The Federated Query Results service's only purpose is to expose information about, and management of, these Resource instances. This Resource contains the current status of the query it corresponds to, any exceptions which occurred during processing, and eventually the results of the query. It supports standard WSRF Resource Lifetime behavior. As such, it exposes, as Resource Properties, the current time (as believed by the local system), and the termination time of the Resource. Once created, the resource will be terminated/destroyed by the service once its termination time is past. This lifetime is initially controlled by a setting in the grid service. The client can also immediately destroy the resource with the `Destroy` operation, or change its termination time with the `SetTerminationTime` operation. Both of these operations are standardized operations for resources supporting Resource Lifetime and as such corresponding common Resource Lifetime clients may be used (though the Federated Query Results client API also makes these operations available).

In addition to the operations and resource properties necessary to support Resource Lifetime on the resource, the service also provides the `getResults` and `isProcessingComplete` operations. The `isProcessingComplete` operation returns a simple Boolean value, indicating whether or not the query processing has completed. Once the query processing has completed, the results can be accessed via the `getResults` operation, which returns a `DCQLQueryResultsCollection`. If the operation is invoked prior to the processing being complete, a `ProcessingNotCompleteFault` fault will be thrown. If the processing is complete, but an exception occurred, a `FederatedQueryProcessingFault` will be thrown, and its cause will be the exception that occurred during query processing.

Security Considerations

The Federated Query Processor services support two deployment scenarios. The first is an insecure deployment, wherein no security (authentication or authorization) is enforced by the container. In this scenario, no encryption is used and no protection of query results is enforced. That is, anonymous communication is used over an open channel, and it is possible for one client to manipulate the query resources of another, given it knows the EPR.

The recommended second scenario is when the services are deployed securely, such as with transport level security (https). Deployments using transport level security ensure integrity and privacy of the communication channel (and obviously the data traveling over it). In this scenario, no authorization is performed by the query service, but any query results created via asynchronous queries are protected such that only the issuer of the query can view or manipulate the results. In this scenario, clients should use credentials to ensure proper protection of query results.

The services do not make use of delegated credentials, and as such, remote data services are accessed either with an identification or anonymously, or with the Federated Query Processor's credentials (depending on the deployment scenario and the settings of the remote data services). While future work may enable this feature,

clients needing this capability (credentialed access of data services) now may leverage the federated query engine directly using the API.

API Details

gov.nih.nci.cagrid.fqp.client.FederatedQueryProcessorClient

The `FederatedQueryProcessorClient` is the main client interface to the federated query service (Figure 6-4).

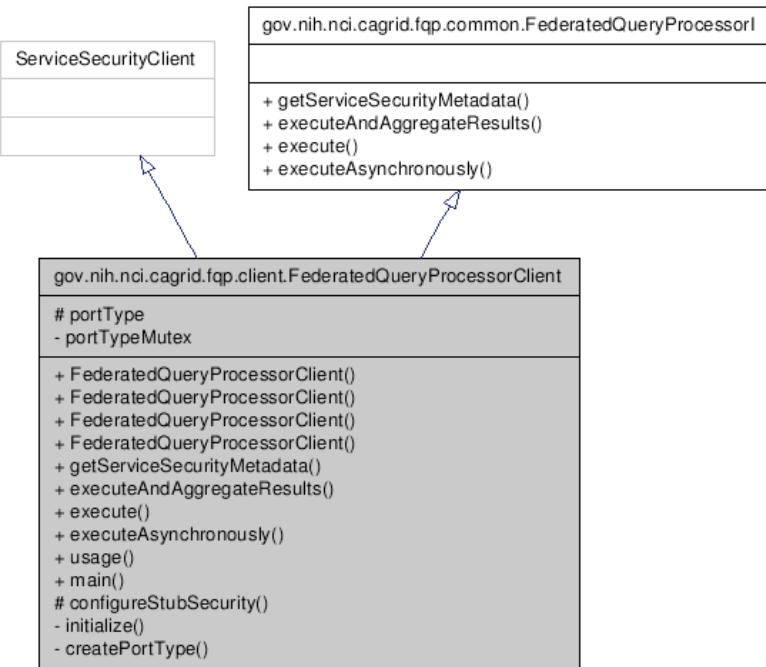


Figure 6-4 *FederatedQueryProcessor Inheritance Model*

Constructor Documentation

`FederatedQueryProcessorClient (String url)`

throws `MalformedURIException, RemoteException`

`FederatedQueryProcessorClient (String url, GlobusCredential proxy)`

throws `MalformedURIException, RemoteException`

`FederatedQueryProcessorClient (EndpointReferenceType epr)`

throws `MalformedURIException, RemoteException`

`FederatedQueryProcessorClient (EndpointReferenceType epr, GlobusCredential proxy)`

throws `MalformedURIException, RemoteException`

Member Function Documentation

Access to caGrid Service Security Metadata:

- gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata
getServiceSecurityMetadata ()
 - throws RemoteException
 - **Description:** Returns the caGrid service security metadata

Distributed Query Methods:

- gov.nih.nci.cagrid.cqlresultset.CQLQueryResults
executeAndAggregateResults (gov.nih.nci.cagrid.dcql.DCQLQuery query)
 - throws RemoteException,
gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault
 - **Description:** Executes the DCQL query, aggregating and returning them as standard data service results
- gov.nih.nci.cagrid.dcqlresult.DCQLQueryResultsCollection execute
(gov.nih.nci.cagrid.dcql.DCQLQuery query)
 - throws RemoteException,
gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault
 - **Description:** Executes the DCQL query, aggregating and returning them as standard data service results
- gov.nih.nci.cagrid.fqp.results.client.FederatedQueryResultsClient
executeAsynchronously (gov.nih.nci.cagrid.dcql.DCQLQuery query)
 - throws RemoteException,
org.apache.axis.types.URI.MalformedURIException
 - **Description:** Executes the DCQL query asynchronously, returning a results client which can be used to access the results

gov.nih.nci.cagrid.fqp.results.client.FederatedQueryResultsClient



Figure 6-5 FederatedQueryResultsClient Inheritance Model

Constructor Documentation

```
FederatedQueryResultsClient (String url)
throws MalformedURIException, RemoteException

FederatedQueryResultsClient (String url, GlobusCredential proxy)
throws MalformedURIException, RemoteException

FederatedQueryResultsClient (EndpointReferenceType epr)
throws MalformedURIException, RemoteException

FederatedQueryResultsClient (EndpointReferenceType epr, GlobusCredential proxy)
throws MalformedURIException, RemoteException
```

Member Function Documentation

Access to caGrid Service Security Metadata

- gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata
 getServiceSecurityMetadata ()
 - throws RemoteException
 - Description: Not used.

Resource Lifetime Methods

org.oasis.wsrf.lifetime.DestroyResponse **destroy** (org.oasis.wsrf.lifetime.Destroy params)

throws RemoteException

Description: Destroys the corresponding resource and query results.

org.oasis.wsrf.lifetime.SetTerminationTimeResponse **setTerminationTime** (org.oasis.wsrf.lifetime.SetTerminationTime params)

throws RemoteException

Description: Sets the time at which the resource and corresponding query results should be destroyed.

Query Results Methods

gov.nih.nci.cagrid.dcqlresult.DCQLQueryResultsCollection **getResults** ()

throws RemoteException,

gov.nih.nci.cagrid.fqp.results.stubs.types.ProcessingNotCompleteFault,

gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault,

gov.nih.nci.cagrid.fqp.stubs.types.InternalErrorFault

Description: Returns the query results, if processing is complete. If processing is not complete, throws ProcessingNotCompleteFault. If processing is complete, but threw an exception, that exception is then rethrown.

```
boolean isProcessingComplete ()
throws RemoteException
```

Description: Returns true if and only if processing of the query is complete.

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine

Constructor Documentation

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.FederatedQueryEngine ()

Default constructor

Member Function Documentation

DCQLQueryResultsCollection

**gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.execute (DCQLQuery
dcqlQuery) throws FederatedQueryProcessingException**

Call Federated Query Processor, and send the generated CQLQuery to each targeted service, placing each result into a single DCQLQueryResults object.

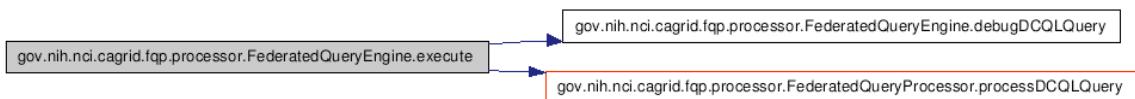
Parameters:

dcqlQuery

Returns:

Exceptions:

FederatedQueryProcessingException



CQLQueryResults

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.executeAndAggregateResults (DCQLQuery dcqlQuery) throws FederatedQueryProcessingException

Call Federated Query Processor, and send the generated CQLQuery to each targeted service, aggregating the results into a single CQLQueryResults object.

Parameters:

dcqlQuery

Returns:

Exceptions:

FederatedQueryException



API Usage Examples

The examples below show various uses of the federated query infrastructure; both executing the grid services and the engine's stand alone API. The exception handling is omitted from the examples for demonstration purposes. Additionally, the result processing shown is simplistic. The results returned from the federated query APIs, `DCQLResultsCollection` instances, are simple container objects for standard Data Service results, `CQLQueryResults`. A `DCQLResult` is just a `CQLQueryResult` and the address of the service which created the result. Additional details on how Data Service results can be processed can be found in the Data Service section of this document.

Executing a Blocking Query

The first example, shown below in Figure 6-6, demonstrates the simplest case of executing a query using the Federated Query Processor grid service, via the `FederatedQueryProcessorClient`. An instance of the client is constructed on lines 1 and 2, wherein the address of the service is provided. Next, a query document is created by loading it from a file (in this case the "exampleDCQL.xml" file) on the local file system. The caGrid common utilities are used to accomplish this on lines 4 and 5. Though not shown, DCQL queries can also be constructed programmatically, just as CQL queries can. On line 7, the service is requested to perform the query as described in the `DCQLQuery`; this operation will block until the processing is complete. The result of the invocation is a `DCQLResultsCollection`, which contains an array of `DCQLResult`. Lines 10-16 loop over these results, and then print out the URL of the service which yielded the result and the number of data instances it produced.

```

1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2       "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5       "exampleDCQL.xml", DCQLQuery.class);
6
7   DCQLResultsCollection dcqlResultsCol = queryClient.execute(dcql);
8   DCQLResult[] dcqlResults = dcqlResultsCol.getDCQLResult();
9   if (dcqlResults != null) {
10      for (int i = 0; i < dcqlResults.length; i++) {
11          DCQLResult result = dcqlResults[i];
12          String targetServiceURL = result.getTargetServiceURL();
13          System.out.println("Got results (" +
14              + result.getCQLQueryResultCollection()
15              .getObjectType().length + ") from:" + targetServiceURL);
16      }
17  } else {
18      System.out.println("Got no results.");
19  }

```

Figure 6-6 Executing a Distributed Query

Executing a Non-Blocking Query

The next example, shown in Figure 6-7, illustrates the capability of the federated query service infrastructure to execute queries in the background, and allow clients to process the results later (suitable for long running queries, or distributed processing of results). It begins to diverge from the example above when it invokes `executeAsynchronously` on line 8 (as opposed to `execute`, as shown in Figure 6-6). This operation returns a new instance of the `FederatedQueryResultsClient`, which can be used to access the results once they are completed. This instance is returned before the service actually starts processing the query. The `FederatedQueryProcessorClient` abstracts the details that an EPR was actually returned by the service, and it directly provides the caller the appropriate API to communicate with the stateful service container for the not yet populated query results. Lines 10-15 demonstrate checking the status of the processing and printing out a “.” to the screen each second until the processing is complete. Future versions of the service will support subscriptions and notifications of query status and completion. The `isProcessingComplete` operation will return false until the query results are available, or the processing is terminated by failure. Once processing is complete, the results can be accessed via the `getResults` operation, shown on line 17. This returns a `DCQLQueryResultsCollection`, just as `execute` did in Figure 6-6. At this point, in terms of result processing, there is no longer a difference between asynchronous (this example) and synchronous (the previous example) query logic. However, to further demonstrate how to process the results, the example below loops over each `DCQLResult` (as before), and prints out the service which yielded it (on line 23). It then, on line 24-27, accesses the standard Data Service query result type, and constructs a `CQLQueryResultsIterator` to iterate over each Object result. This iteration, shown on lines 29-33, prints the XML representation of each Object, as the iterator was constructed to use “xml only” by passing true to the second argument of its constructor on line 27.

```

1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2     "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5     "exampleDCQL.xml", DCQLQuery.class);
6
7   FederatedQueryResultsClient resultsClient =
8     queryClient.executeAsynchronously(dcql);
9
10  System.out.print("Waiting for completion");
11  while (!resultsClient.isProcessingComplete()) {
12    Thread.sleep(1000);
13    System.out.print(".");
14  }
15  System.out.println("\nProcessing Complete.");
16
17  DCQLQueryResultsCollection dcqlResultsCol = resultsClient.getResults();
18  DCQLResult[] dcqlResults = dcqlResultsCol.getDCQLResult();
19  if (dcqlResults != null) {
20    for (int i = 0; i < dcqlResults.length; i++) {
21      DCQLResult result = dcqlResults[i];
22      String targetServiceURL = result.getTargetServiceURL();
23      System.out.println("Got results from:" + targetServiceURL);
24      CQLQueryResults queryResultCollection =
25        result.getCQLQueryResultCollection();
26      CQLQueryResultsIterator iterator =
27        new CQLQueryResultsIterator(queryResultCollection, true);
28      int resultCount = 0;
29      while (iterator.hasNext()) {
30        System.out.println("===== RESULT [" + resultCount++ + "] =====");
31        System.out.println(iterator.next());
32        System.out.println("===== END RESULT=====\\n\\n");
33      }
34    }
35  }

```

Figure 6-7 Executing a Non-Blocking Distributed Query

Removing Query Results

As mentioned above, the Federated Query Results grid service is a stateful WSRF service, and executing asynchronous queries with the Federated Query Processor service generates Resources on it. These resources house the query results for clients, and can be accessed multiple times. The example shown below in Figure 6-8, demonstrates how, after the client is done with the results, they should be removed from the service by invoking the destroy operation of the `FederatedQueryResultsClient`, shown on line 12. The results will eventually “expire” and be automatically removed after a service-specified lifetime, unless their lifetime is extended as shown in Figure 6-9., but it is good practice to manually release unneeded resources.

```

1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2     "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5     "exampleDCQL.xml", DCQLQuery.class);
6
7   FederatedQueryResultsClient resultsClient =
8     queryClient.executeAsynchronously(dcql);
9
10 // ... Process the results ...
11
12 resultsClient.destroy(new Destroy());

```

Figure 6-8 Destroying Query Results

Scheduling Removal of Query Results

As mentioned above, query results will expire after a service-configure default lifetime. However, this can be controlled by the client by specifying a new termination time for the resource. The example shown below in Figure 6-9, demonstrates how one client may request the service to perform a federated query, set the lifetime of the yet to be created results, and hand off an EPR to those results to some other processing thread or client. In this example, the query is executed asynchronously just as before (on lines 7-8). However, rather than waiting on completion of the results, the example uses, on line 13, the setTerminationTime operation of the results client to request the results live on the service for four hours. This is accomplished by creating a Resource Lifetime standard SetTerminationTime request, and providing it a Calendar instance configured to be four hours later than the current time, as shown on lines 11 and 12. The result of this operation is a Resource Lifetime standard SetTerminationTimeResponse, which indicates the current time, as believed by the service, and the time at which the resource will be destroyed. The example prints these out, as well as the value of the local Calendar instance, on lines 18-23. Finally, the code demonstrates, on line 25, how an EPR to the result Resource can be accessed from the client with the getEndPointReference method.

It should be noted that if the client's system clock and the service's system clock have significant differences in their belief of the current time, this example code could cause the resource to terminate either earlier or later than expected. In cases where this may be an issue for clients, the results service provides as a Resource Property, the current time as believed by the service. A client could access this resource property, and construct the Calendar instance from it, rather than using its local system clock.

```
1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2     "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5     "exampleDCQL.xml", DCQLQuery.class);
6
7   FederatedQueryResultsClient resultsClient =
8     queryClient.executeAsynchronously(dcql);
9
10  SetTerminationTime termTime = new SetTerminationTime();
11  Calendar terminateAt = Calendar.getInstance();
12  terminateAt.add(Calendar.HOUR, 4);
13  termTime.setRequestedTerminationTime(terminateAt);
14
15  SetTerminationTimeResponse response =
16    resultsClient.setTerminationTime(termTime);
17
18  System.out.println("Service's current time "
19    + response.getCurrentTime().getTime());
20  System.out.println("Requested termination time "
21    + terminateAt.getTime());
22  System.out.println("Scheduled termination time "
23    + response.getNewTerminationTime().getTime());
24
25  EndpointReferenceType resultEPR = resultsClient.getEndpointReference();
26  // ... Hand off result EPR to some other process or client ...
```

Figure 6-9 Scheduling the Destruction of Query Results

Using the Engine Directly to Access Protected Data Services

The final example, shown below in Figure 6-10, demonstrates how the federated query infrastructure can be used to locally execute federated queries (as opposed to requesting the service to execute them). While most clients will opt to use the service interfaces, there are many reasons why a client may wish to invoke the engine locally, such as minimizing data movement. The most common reason to use the engine locally is if the Data Services being targeted requires authorization to access the data of interest. The federated query processor service infrastructure does not currently have the capability to assume the identity of the client requesting the query; it queries Data Services either anonymously, or with its own credentials (depending on deployment scenarios). When executing the engine locally, the client has the ability to use credentials when it queries data services. When executing locally, the engine will make use of the “default” Globus credentials if the Data Service being accessed does not allow anonymous access. Consult the caGrid security documentation on how to get and set default Globus credentials.

Using the engine locally is similar to using the processor service. The engine can be constructed, as shown on line 1, by instantiating a `FederatedQueryEngine`. This has the same query execution methods as the grid service, except it does not provide asynchronous execution (as this is easily done locally). The example below (Figure 6-10) shows the ability of the engine to aggregate DCQL results and return them as standard Data Service Results, shown on line 4. It then uses the `CQLQueryResultsIterator` to print the XML representation of the results, just as in Figure 6-7.

```
1  FederatedQueryEngine fqp = new FederatedQueryEngine();
2  DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument("exampleDCQL.xml",
3      DCQLQuery.class);
4  CQLQueryResults results = fqp.executeAndAggregateResults(dcql);
5  CQLQueryResultsIterator iterator = new CQLQueryResultsIterator(results, true);
6  int resultCount = 0;
7  while (iterator.hasNext()) {
8      System.out.println("=====RESULT [" + (resultCount++) + "] =====");
9      System.out.println(iterator.next());
10     System.out.println("=====END RESULT=====\\n\\n");
11 }
```

Figure 6-10 Invoking the FederatedQueryEngine Locally

Chapter 7 WS-Enumeration

This chapter describes the client-side APIs for enumerations.

Topics in this chapter include:

- *Overview* on this page
- *Utilizing Enumeration in a Grid Service* on page 164
- *Client API* on page 166
- *Examples* on page 169

Overview

CaGrid provides mechanisms for integration of the WS-Enumeration specification both singularly and as part of the Bulk Data Transfer support. This integration is accomplished with a variety of provided tools for both client and server side use, as well as a specialized service extension to the Introduce toolkit.

An overview of WS-enumeration is beyond the scope of this document. For more information, see the following websites:

- The [The WS-Enumeration spec](#) document is available from Globus
- [WS-Enumeration project at dev.globus.org](#)

Enumeration Service Context

The WS-Enumeration spec makes the assumption that the service context which provides a method to begin an enumeration is also the same context which implements the other enumeration methods (Pull, Renew, Status, etc). In caGrid, the data to be enumerated is stored in a server side WSRF resource. This means the method to begin an enumeration must return both the spec required Enumeration Context and an Endpoint Reference (EPR) indicating the service context which will be responsible for handling the enumeration, as well as the resource key for the data. In caGrid, this combined response is known as an Enumeration Response Container.

When WS-Enumeration is utilized from within a Bulk Data Transfer resource, the BDT resource creates the enumeration resource and returns the Enumeration Response Container to encapsulate both the EPR of the new service context and resource, and the enumeration context object.

Extension Implementation

caGrid's support for WS-Enumeration is provided by a service extension to the Introduce toolkit. This extension adds the service context for WS-Enumeration operations, copies relevant WSDL files and schemas, and finally sets the Globus provided `EnumProvider` class as the implementation for the enumeration operations

Utilizing Enumeration in a Grid Service

To utilize WS-Enumeration in a grid service, two things are required. First, the service must be generated with the caGrid WS-Enumeration extension enabled. Next, one or

more operations must be added to the grid service which return an Enumeration Response Container.

Such a "begin enumeration" method is required to begin the enumeration and hand off control of some data resource to the enumeration context and resource. The WS-Enumeration resource requires an implementation of the `org.globus.ws.enumeration.EnumIterator` interface be supplied to it at creation time. This interface is the means through which a data resource is exposed with the server side enumeration implementation. To simplify the process of exposing data via this interface, the caGrid enumeration implementation includes several utilities.

Enumeration Implementations

caGrid WS-Enumeration supplies the factory class

`gov.nih.nci.cagrid.wsenum.utils.EnumIteratorFactory` which creates concrete instances of the `EnumIterator`. The factory method `createIterator()` takes parameters to determine the type of implementation, as well as the list of data objects to be enumerated (or a `java.util.Iterator` to the same), the XML QName of those objects, and a WSDD configuration stream to manage the serialization of the data objects. The Java enumeration type `gov.nih.nci.cagrid.wsenum.utils.IterImplType` defines the five implementations and gives a brief description of each:

- `GLOBUS_SIMPLE`
 - The Globus-provided simple enum iterator.
- `GLOBUS_INDEXED_FILE`
 - The Globus-provided indexed file enum iterator.
- `CAGRID_SIMPLE`
 - A simple iterator which persists objects to disk.
 - This iterator only respects the `maxElements` iteration constraint.
- `CAGRID_THREADED_COMPLETE`
 - This iterator uses threads to respect `maxTime` constraints as well as respecting `maxCharacters`. Elements overflowing either of these constraints, however, are lost, and wait states for thread completion are not optimized.
- `CAGRID_CONCURRENT_COMPLETE`
 - This iterator uses the Java 5 `java.util.concurrent` package to fully support the WS-Enumeration specification for an `EnumIterator` implementation. All iteration constraints are respected, and elements which cause `maxCharacters` to be exceeded are queued for later retrieval.

For most purposes, the `CAGRID_CONCURRENT_COMPLETE` implementation should be used as provides full support for the server side WS-Enumeration spec. This is also the default implementation selected by the caGrid data services infrastructure. The other implementations are less complete, and may be useful in emulating the behavior of other enumeration-enabled systems.

Creating an Enumeration Response

Once an `EnumIterator` instance has been created, it must be sent to the enumeration resource, and an Enumeration Response Container returned to the user. The caGrid

provided utility class `gov.nih.nci.cagrid.wsenum.utils.EnumerateResponseFactory` encapsulates this functionality in a simple static method. The method `createEnumerationResponse()` takes a single `EnumIterator` parameter and returns an **Enumeration Response Container**, which can immediately be returned to the client. This method encapsulates locating the `EnumResourceHome`, creating a new resource, setting its visibility, and deriving an endpoint reference (EPR) from the resulting resource key.

Example

Below is an example code snippet which uses the provided utilities to create an enumeration iterator over a list of Strings and return an appropriate enumeration response container:

```
// create a list of data values
List<String> values = new ArrayList<String>();
for (int i = 0; i < 10; i++) {
    values.add("String Value " + i);
}

// create an enum iterator
EnumIterator enumIter = EnumIteratorFactory.createIterator(
    IterImplType.CAGRID_CONCURRENT_COMPLETE,
    values,
    new QName("http://www.w3.org/2001/XMLSchema", "string"),
    null);

// formulate the response container and return
EnumerationResponseContainer response =
    EnumerateResponseFactory.createEnumerationResponse(enumIter);
return response;
```

Client API

The Globus-provided `org.globus.ws.enumeration.ClientEnumIterator` API provides `java.util.Iterator` abstraction for retrieving enumeration data and supports automatic data deserialization. The caGrid WS-Enumeration implementation provides a simplified factory interface to create a new instance of a Client Enum Iterator from an Enumeration Response Container.

For a more detailed discussion of the WS-Enumeration client tools, please see the [Globus developer's wiki regarding the WS Core WS-Enumeration](#).

ClientEnumeration

The ClientEnumeration API provides basic functions for managing enumeration lifetime and retrieving its data. ClientEnumeration must be initialized with a `javax.xml.rpc.Stub` instance that is a Stub for the service that implements the WS-Enumeration operations and with an `EnumerationContextType` object returned by the `enumerate` operation of the service or any other operation that initiates an enumeration.

The `javax.xml.rpc.Stub` instance must define all of the WS-Enumeration operations except the `enumerate` operation. Also, the `Stub` instance must be properly configured with the security properties if calling a secure service.

IterationResult pull(IterationConstraints constraints)

Retrieves the next set of elements of the enumeration. The input parameter defines the constraints for the operation such as the maximum number of elements to retrieve, the maximum number of characters that the consumer can accept, and the maximum amount of time in which the data needs to be returned. The return parameter contains the results of the iteration and an end of sequence flag to indicate if there is more data to be returned. The results of the iteration are of the `javax.xml.soap.SOAPElement` type.

This method calls the WS-Enumeration `pull` operation on the data service.

IterationResult pull()

Same as `pull(IterationConstraints)` function but uses default constraints (maximum number of elements set to 1, no maximum characters limit and no time limit).

void release()

Explicitly releases the enumeration. In general, the enumeration context is automatically released when a client finishes retrieving all the enumeration data or the enumeration expires if it was configured with an expiration time or duration. In cases where no expiration time was set for the enumeration or when not enumerating over the entire data, the enumeration should be released explicitly.

This method calls the WS-Enumeration `release` operation on the data service.

EnumExpiration renew(EnumExpiration expiration)

Sets a new expiration time/duration of the enumeration. The input parameter can be null to configure the enumeration without an expiration time/duration (the enumeration will not expire). The expiration time/duration cannot be in the past (as according to the service clock). The service can choose to accept a different expiration time than specified. The return parameter can also be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration `renew` operation on the data service.

EnumExpiration getStatus()

Gets the current expiration time/duration of the enumeration. The return parameter can be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration `getStatus` operation on the data service.

ClientEnumIterator

The `ClientEnumIterator` API provides a simple-to-use API for enumerating over data using the WS-Enumeration operations. The `ClientEnumIterator` class implements the `java.util.Iterator` interface but the implementation of these functions does not follow the `Iterator` contract exactly because of the WS-Enumeration specification limitations. The `ClientEnumIterator` class uses the `ClientEnumeration` API underneath and in contrast to the `ClientEnumeration` API offers automatic data deserialization.

The `ClientEnumIterator` must be initialized with a `javax.xml.rpc.Stub` instance that is a Stub for the service that implements the WS-Enumeration operations and with an

`EnumerationContextType` object returned by the `enumerate` operation of the service or any other operation that initiates an enumeration.

The `javax.xml.rpc.Stub` instance must define all of the WS-Enumeration operations except the `enumerate` operation. Also, the Stub instance must be properly configured with the security properties if calling a secure service.

During iteration the `ClientEnumIterator` makes remote calls to the data service to retrieve the next set of items (calls the WS-Enumeration `pull` operation). The frequency of these remote calls is controlled by the `maxElements` setting of the `IterationConstraints` of the `ClientEnumIterator`. If that number is small the `ClientEnumIterator` will make a lot of remote calls but will use a small amount of memory (since it always keeps a few of the items in the memory). But if that number is big the `ClientEnumIterator` will make fewer remote calls but will use more memory (since it now keeps more items in the memory).

void setItemType(Class itemType)

Sets the type of the object returned by the `next()` operation. By default the objects returned by the `next()` operation will be of the `javax.xml.soap.SOAPElement` type. If the item type is set, the enumeration elements will be automatically deserialized into this type. This assumes the enumeration elements are all of the same type.

void setIterationConstraints (IterationConstraints constraints)

Sets iteration constraints for the iterator. By default the constraints are not set and defaults are assumed (maximum number of elements set to 1, no maximum characters limit and no time limit).

Object next()

Returns the next object in the enumeration. The returned object can be null. If item type is set (using the `setItem Type` method) the current object will be automatically converted into that type and returned. Otherwise, an object of the `javax.xml.soap.SOAPElement` type is returned. If the enumeration has ended (`hasNext()` returns `false`) or has been released on the client, the `NoSuchElementException` is raised. Also, in certain cases the `NoSuchElementException` can also be raised even though `hasNext()` is returned as `true`. If deserialization is performed and it fails, a `ConversionException` is raised and the index of the iteration is not advanced (so that the user can specify another item type or disable deserialization).

This method calls the WS-Enumeration `pull` operation on the data service.

boolean hasNext()

Tests if there are more elements in the enumeration to be returned. If it returns `false`, there are no more elements to be returned. If `true`, there **might** be more elements to be returned. This method can return `true` even though the `next()` operation consistently returns null. Also, this method can return `true` even though the `next()` operation will throw `NoSuchElementException`.

Object convert(SOAPElement element)

Performs object conversion on the enumeration element. This function is called by the `next()` function every time the `next()` function is called. It can be used to deserialize the enumeration element into the appropriate Java object. This function is meant to be overwritten by the subclasses of the `ClientEnumIterator` to provide custom deserialization behavior. If deserialization fails, then a `ConversionException` is thrown.

void release()

Explicitly releases the enumeration context.

This method calls the WS-Enumeration *release* operation on the data service. `hasNext()` will return `false` and `next()` will throw `NoSuchElementException` after this method is called.

Utilities

The caGrid-provided class

`gov.nih.nci.cagrid.wsenum.utils.EnumerationResponseHelper` contains static methods which can take an Enumeration Response Container and return a client enum iterator instance. The method `createClientIterator` takes only the response container, while another implementation of this method takes both the container and a `java.io.InputStream` to the `client-config.wsdd` file. The information contained in this file will be used to deserialize results from the enumeration.

Examples

Basic caGrid WS-Enumeration

Given a service (here an enumeration-enabled Data Service) which supports enumeration, the following pattern may be used by a client to enumerate over data results.

```
TestEnumerationDataServiceClient client = new
TestEnumerationDataServiceClient(args[1]);

EnumerationResponseContainer response = client.enumerationQuery(null);

ClientEnumIterator iter = EnumerationResponseHelper.createClientIterator(
    response,
TestEnumerationDataServiceClient.class.getResourceAsStream("client-
config.wsdd"));

// optional use of iteration constraints to fetch 10 elements w/o regard
// to number of characters or maximum time to wait
IterationConstraints cons = new IterationConstraints(10, -1, null);
iter.setIterationConstraints(cons);

while (iter.hasNext()) {
    SOAPElement elem = (SAPElement) iter.next();
    if (elem != null) {
        // handle the data element
    }
    // optionally change constraints to fetch 5 elements, max of
    // 50000 characters, and a max wait time of 1 hr, 30 minutes
    cons = new IterationConstraints(5, 50000,
        new Duration(false, 0, 0, 0, 1, 30, 0));
}
```

```
    iter.setIterationConstraints(cons);  
}
```

The use of IterationConstraints to express how data should be fetched from the service is optional, and the default behavior is to retrieve one element at a time, with no regard for max characters or time constraints. Some server side implementations do not respect all iteration constraints. These constraints may change at any time during enumeration.

While iterating, it is possible that the call to `hasNext` may return true, yet the call to 'next' return null, or even throw a `NoSuchElementException`. This is often the case when results are being dynamically populated on the server side at a rate slower than the client is capable of consuming them. This condition may also arise when the iteration constraints are such that no element could be returned which fulfills the constraints, yet the server is still holding more elements. Changing the constraints (allowing more time, increasing the maximum number of characters) may allow these elements to be retrieved.

Using BDT

The case of using the caGrid Bulk Data Transport infrastructure slightly complicates the client code to access data via enumeration by adding a level of indirection as the BDT resource is created first, and then used to create an enumeration resource.

```
TestBDTDataServiceClient client = new TestBDTDataServiceClient(args[1]);  
  
// query with BDT  
BulkDataHandlerReference bdtRef = client.bdtQuery(null);  
  
// use the reference to create a generic BDT client  
BulkDataHandlerClient bdtClient = new  
BulkDataHandlerClient(bdtRef.getEndpointReference());  
  
// start the enumeration on the server  
EnumerationResponseContainer response = bdtClient.createEnumeration();  
  
// rest of code the same as basic WS-Enumeration case
```

Chapter 8 Workflow Management Service

This chapter describes the architecture and APIs for interacting with caGrid workflow.

Topics in this chapter include:

- *Overview* on this page
- *Workflow Architecture* on this page
- *WorkflowFactoryService API* on page 172
- *WorkflowServiceImpl API* on page 175
- *Security in WorkflowFactory and Context Services* on page 178
- *Service Selection* on page 178
- *Provenance Tracking* on page 179
- *WS-RF Resources in Workflows* on page 179

Overview

caBIG aims to bring together disparate data and analytic resources into a “World Wide Web of cancer research”. This will be achieved through common standards and software frameworks for the federation of these resources into “grid” services. Many of the tasks in the collection and analysis of cancer-related data on the grid involve the use of workflow. Here, we define workflow as the connecting of services to solve a problem that each individual service could not solve. caGrid implements workflow by providing a grid service for submitting and running workflows that are composed of other grid services.

Workflow Architecture

The workflow component leverages the same infrastructure stack as the caGrid toolkit (GT4, Tomcat, Java, Ant, and Introduce) with the addition of the ActiveBPEL workflow engine. The WorkflowFactoryService is a standard Introduce-built grid service that allows a workflow to be created from a BPEL workflow document. An EPR is returned to a WorkflowImplService resource that can be used to start, stop, pause, resume, cancel, and destroy the created workflow. The WorkflowImplService is layered on top of the ActiveBPEL workflow engine, which provides the primary functionality for running the BPEL-defined workflow. See Figure 8-1 for an overview of this architecture.

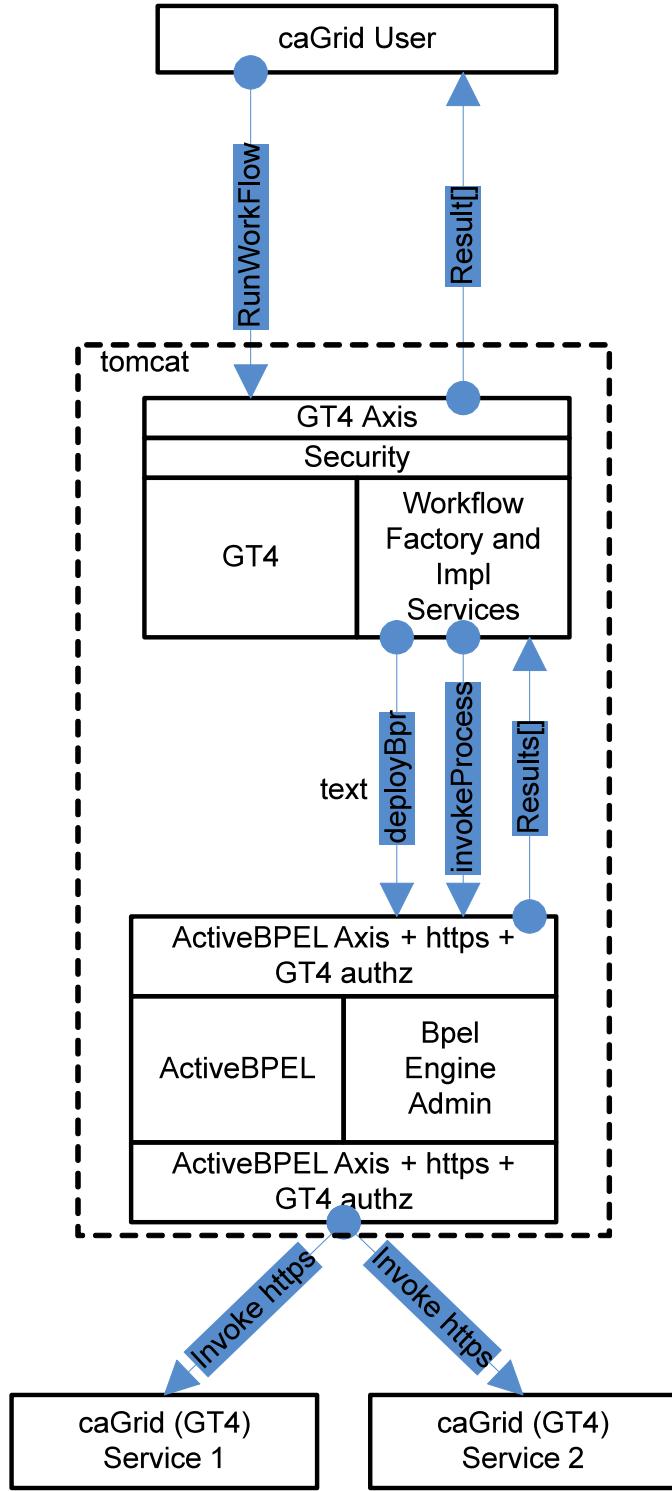


Figure 8-1 Overview of the architecture of the caGrid Workflow component

The following is a walk-through of a user's interactions and the corresponding operations on the Workflow Services based on Figure 8-1:

- User creates a BPEL document that orchestrates one or more caGrid services.

- User configures (if it is not already configured), the workflow Submission GUI to contact the appropriate WorkflowFactory Service endpoint.
- User uses the WorkflowSubmission GUI to submit the BPEL document to a Workflow Factory service. The input BPEL document is parsed and an exception is thrown if it is not well-formed (with respect to schema compliance).
- The factory service implementation invokes PDDGenerator to generate the deployment descriptor for the workflow.
- The factory service implementation invokes BPRGenerator to generate the BPELArchive (called bpr hereafter) that is ready to be deployed.
- The workflow services are configured with the location of ActiveBPEL admin service location at deployment time.
- The factory service implementation invokes deployBpr operation on the ActiveBPEL Admin service, which is a vanilla Axis-based Web Service, to deploy the workflow archive that is created in the previous step.
- The admin service responds with a deployment summary reporting success if the workflow is deployed successfully along with a processId.
- The workflow factory service creates a Workflow Resource that consists of the BPEL document the user submitted. It constructs an EPR for the workflow resource using the processId returned by ActiveBPEL Admin Service and the endpoint url for WorkflowImpl Service that is used to manage the Workflow.
- Once the workflow is deployed successfully, it is deployed as a web service inside ActiveBPEL.
- The user provides the input to the workflow using the Workflow Submission GUI and invokes the start operation on the WorkflowImpl Service.
- The input arguments to the workflow are declared as an array of xsd:any. They are parsed and cast to the types that they are meant to be.
- To start the workflow, the `WorkflowImpl` Service sends a message to the receiving partnerLink in the workflow.
- After the workflow successfully executes, the results are returned to the client app/user by a call to `getWorkflowOutput`.
- The user can invoke the `getStatus` operation on the `WorkflowImpl` Service using the Workflow GUI to find out the status of the Workflow Execution. The workflow status is returned in one of the five possible states that a workflow can take (Submitted, Active, Finished, Failed, Suspended).
- If more detailed status is desired, `getDetailedStatus` operation can be invoked through the workflow gui which provides the user the node that is currently being executed in the workflow.

WorkflowFactoryService API

Workflows are created using the `WorkflowFactoryService`, which is a grid service that follows the resource pattern. The returned object holds an EPR to a `WorkflowImplService`, which can be used to manipulate the created workflow.

Public WorkflowFactoryOutputType
createWorkflow(WorkflowDescriptionType wmsInputType) throws
WorkflowException ()

Description

This method creates a workflow resource from the BPEL document found in `wmsInputType` and returns an EPR of the created resource to the client. The BPEL resource, along with the most recent state, is persisted in a MySQL database and is recovered in the event of a container crash.

WorkflowDescriptionType

This is the input to `createWorkflow`, and it consists of `workflowName`, a String `bpelDoc`, an Array of `wsdlReferences`, and an initial termination time for the workflow. If the termination time is not specified, the service defaults to 24 hours. Termination of the workflow invalidates the `WorkflowManagementService` EPR and any running workflow is stopped.

```
<xsd:complexType name="WorkflowDescriptionType">
  <xsd:sequence>
    <xsd:element name="workflowName" type="xsd:string" minOccurs="1"
maxOccurs="1" />
    <xsd:element name="bpelDoc" type="xsd:string" maxOccurs="1" />
    <xsd:element name="wsdlReferences" type="tns:WSDLReferences"
maxOccurs="unbounded" />
    <xsd:element name="InitialTerminationTime" type="xsd:dateTime"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WSDLReferences">
  <xsd:sequence>
    <xsd:element name="wsdlNamespace" type="xsd:anyURI"/>
    <xsd:element name="wsdlLocation" type="xsd:string"/>
    <xsd:element name="serviceUrl" type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:complexType>
```

WorkflowFactoryOutputType

This is the output of the `createWorkflow` method. An EPR is constructed by the factory and returned to the client. At this point the workflow document is deployed in the workflow engine and is also stored in a database, but it has not started. The EPR points to an instance of the `WorkflowManagementService`, which should be used to start the workflow.

```
<xsd:complexType name="WorkflowFactoryOutputType">
  <xsd:annotation>
    <xsd:documentation>This type represents the output from a
workflow</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="WorkflowEPR" type="wsa:EndpointReferenceType" />
  </xsd:sequence>
</xsd:complexType>
```

Faults

`UnableToDeployWorkflowFault`: This fault is thrown if the workflow is unable to be deployed (e.g. the BPEL document submitted fails pre-deployment validation).

`InvalidBPELFault` extends `UnableToDeployWorkflowFault`: This fault is thrown if the BPEL document submitted fails pre-deployment validation (e.g. not valid XML).

Factory ResourceProperties

We intend to provide aggregate resource properties on the factory service in our next iteration. Following are some of the examples of those:

- Total number of workflows
- ListOfWorkflowsSubmitted

WorkflowServiceImpl API

This service is used to manage the workflow resources created by the `WorkflowFactoryService` (Figure 8-2). The service provides asynchronous execution of deployed workflows. The following are the operations the service provides in addition to the standard WS-RF operations such as `destroy()`, `setTerminationTime()`, etc.

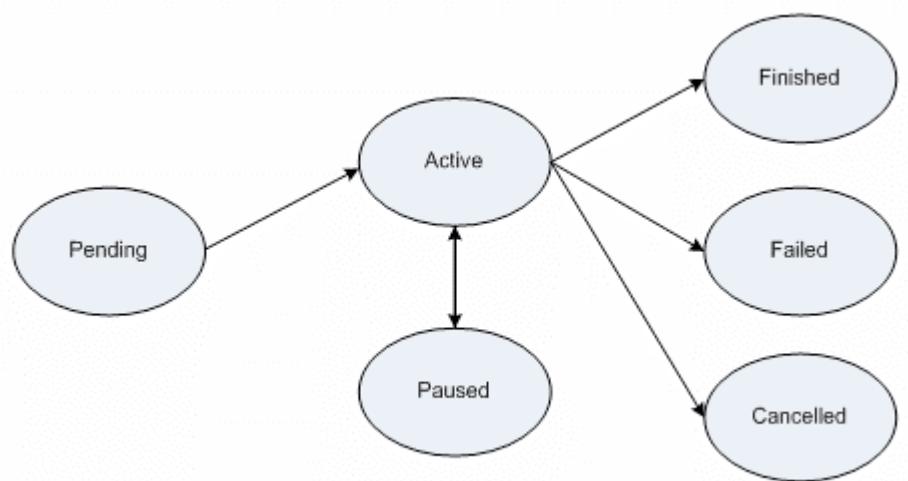


Figure 8-2 Workflow service state diagram

Public `WorkflowStatusType start(StartInputType input)` throws `WorkflowException`, `StartCalledOnStartedWorkflowFault`

Description

This operation is used to start the workflow deployed using the factory with a set of input parameters. The input parameters are modeled as an array of `xsd:any` elements. The output is a void type.

```

<xsd:complexType name="StartInputType">
  <xsd:sequence>
  
```

```

<xsd:element name="inputArgs" type="tns:WorkflowInputType" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WorkflowInputType">
<xsd:sequence>
<xsd:any maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

```

Faults

- `StartCalledOnStartedWorkflowFault`: This is thrown if `start()` is called on a workflow that is not in any one of the terminal states (i.e. Done, Failed, Cancelled).
- `WorkflowException`: Every other fault results in the service throwing this with a message describing more details as to what went wrong.
- `Public WorkflowStatusType getStatus() throws WorkflowException`

Description:

This operation is used to query for the status of the deployed workflow. `WorkflowStatusType` includes a fault.

```

<xsd:simpleType name="WorkflowStatusType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Pending" />
<xsd:enumeration value="Active" />
<xsd:enumeration value="Done" />
<xsd:enumeration value="Failed" />
<xsd:enumeration value="Cancelled" />
</xsd:restriction>
</xsd:simpleType>

```

- `Public WorkflowStatusType pause() throws CannotPauseFault`

Description:

This operation pauses the workflow until `resume()` or `cancel()` is invoked. This operation translates to invoking an equivalent operation provided in the ActiveBPEL Admin interface. When the pause operation is invoked, ActiveBPEL stops the execution of the workflow document which means that there would no further service invocations or other activities. However, this will not affect the invocations in progress when the `pause()` is invoked. This operation returns the new state of the workflow resource (which always should be Active).

- `Public WorkflowStatusType resume() throws CannotResumeFault`

Description:

This operation resumes a paused workflow. It translates to invoking an equivalent operation provided in the ActiveBPEL Admin interface.

- `Public WorkflowOutputType getWorkflowOutput() throws WorkflowException`

Description:

This operation is used to get the final output of a completed workflow. It will return a fault if the workflow is not yet completed. If ActiveBPEL allows for intermediate access of results, then this operation can potentially return the last

result that the workflow engine has for this workflow. The output is modeled as an array of xsd:any elements.

```
<xsd:complexType name="WorkflowOutputType">
  <xsd:sequence>
    <xsd:any maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>
```

- Public void cancel() throws WorkflowException

Description:

This operation terminates a workflow. It translates to invoking an equivalent operation provided in the ActiveBPEL Admin interface.

- Public WorkflowStatusEventType[] getDetailedStatus() throws WorkflowException

Description:

This operation is used to get more detailed status of the submitted workflow. ActiveBPEL provides a web application to find more detailed status of the submitted workflows. However, exposing this management interface to end users is not advisable as it provides access to more powerful operations like stopping the Workflow Engine, stopping execution of other workflows etc. In this implementation, an attempt is made to provide the same level of status information in the Workflow Submission GUI without using the ActiveBPEL Admin webapp. ActiveBPEL provides an option for executing BPEL processes to log their current state to a file. A file is created for each execution of the BPEL process under \$USER_HOME/AeBPELEngine/process-logs/*.log. This operation does not take any input arguments and the output type is modeled as follows :

```
<xsd:complexType name="WorkflowStatusEventType">
  <xsd:sequence>
    <xsd:element name="timestamp" type="xsd:string"/>
    <xsd:element name="state" type="tns:WorkflowStateType"/>
    <xsd:element name="currentOperation" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Resource Properties:

WorkflowStatusRP:

The status of a workflow is exposed as a Rbsource Property so clients can subscribe to it and get notified when a state change happens. WorkflowStatusType is modeled as an Enum of Strings with the following valid values:

- Pending (Created but Start has not been called)
- Active
- Done
- Paused
- Failed

The status also includes the latest fault a workflow execution throws.

```
<xsd:simpleType name="WorkflowStatusType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Pending" />
    <xsd:enumeration value="Active" />
    <xsd:enumeration value="Done" />
    <xsd:enumeration value="Failed" />
    <xsd:enumeration value="Cancelled" />
    <xsd:enumeration value="Paused" />
  </xsd:restriction>
</xsd:simpleType>
```

WorkflowStartTimeRP:

This property denotes the time when the start() operation is called on the resource.

WorkflowEndTimeRP:

This property denotes the time when the workflow status is set to Done/Failed/Cancelled.

Public void destroy()**Description:**

This is a standard WS-RF operation but is mentioned here to clarify the semantics and what it means to a Workflow Resource. If called, this method will delete a Workflow resource from the database along with the intermediate results and subscriptions for notifications. This operation is called by the GT4 framework when the lifetime of a resource is expired. The lifetime is set in the initial create() call in the factory. Internally, destroy() removes all the database entries for a particular workflow resource, all the subscriptions for notifications, and other temporary resources both in memory and on the disk.

Security in WorkflowFactory and Context Services

Two types of deployment patterns for Workflow are needed in regards to security. One deployment scenario would have the factory and the context service running with grid security (using Transport level security and caGrid authorization) and would require a client to present grid credentials to submit and run workflows. Once a workflow resource is created by a user, programmatic GridMap authorization is used to limit access to the resource to the creator of the resource. Delegation of credentials is performed using the delegation service of the Globus Toolkit. This deployment is used to orchestrate workflows that require secure access to any services involved in the workflow. The other deployment does not have any security and is used to orchestrate workflow between unsecured grid services.

Service Selection

This feature is not implemented yet. A Custom invoke handler is written for ActiveBPEL that queries a pre-configured GT4 index service to get the list of services. The query would be based on input and output types of the service invocation. Once a list of service handles is obtained from the index service, the dynamic endpoint for the service invocation is replaced by the first endpoint in the list.

Provenance Tracking

This is out of scope for this component during this release. No provenance tracking is exposed via the workflow component.

WS-RF Resources in Workflows

A BPEL service can involve affecting the state of a WS-RF resource. It is outlined below how one can go about creating WS-RF resources and passing them in a BPEL workflow using a hypothetical factory and instance service.

The partner links define the different services that interact with the BPEL process. The <partnerLinks> section in the code listing below shows the two services that interact with the workflow.

```
<partnerLinks>
  <partnerLink name="rft" partnerLinkType="RFTPortTypeLink"
    partnerRole="RFTPortTypeProvider"/>

  <partnerLink name="RFTFactory" partnerLinkType="RFTFactoryPortTypeLink"
    partnerRole="RFTFactoryPortTypeProvider"/>
</partnerLinks>
```

The partnerLinks here correspond to the RFT service (rft), as well as the factory for that service (RFTFactory). The partner link type and a role name in each partner link identifies the functionality that must be provided by the partner services. That is, the interface (portType) that the partners need to implement.

Variables

The code below shows some of the message variables used by the BPEL process. The variables are defined in terms of WSDL message types, XML Schema simple types, or XML Schema elements. These variables are used in messages exchanged with partner services.

```
<variables>
  ...
  <variable messageType="ns4:CreateResourceRequest"
    name="CreateResourceRequest"/>
  <variable messageType="ns4:CreateResourceResponse"
    name="CreateResourceResponse"/>
  <variable messageType="ns2:StartTransfersRequest"
    name="gsuRequest"/>
  <variable messageType="ns2:StartTransfersResponse"
    name="rftResponse"/>
  <!-- some variables RFT service & factory -->
  <variable messageType="ns5:GetResourcePropertyRequest"
    name="GetResourcePropertyRequest"/>
  <variable messageType="ns5:GetResourcePropertyResponse"
    name="GetResourcePropertyResponse"/>
  <!-- variable for endpoint reference -->
  <variable element="wsa:EndpointReference"
    name="DynamicEndpointRef"/>
</variables>
```

Creating a Web Service Instance

Creating a new Web service resource instance involves making a call to the `createResource` operation of a designated factory service. This is achieved by using BPEL's service invocation mechanism. The `<invoke>` construct allows a BPEL process to invoke a one-way or request-response operation on a `portType` (interface) ordered by a partner service. The code listing below shows the invocation to the `createResource` operation of the RFT factory service.

```
<invoke  
name="InvokeRFTFactory"  
partnerLink="RFTFactory"  
portType="RFTFactoryPortType">  
operation="createResource"  
inputVariable="CreateResourceRequest"  
outputVariable="CreateResourceResponse"  
</invoke>
```

The `invoke` activity which makes a synchronous call to the factory service, contains the `portType` of the operation as well as the `inputVariable` and `outputVariable` variables. If the invocation is successful, the `outputVariable` will contain the endpoint reference of the created instance. Below is an example SOAP message returned after a successful invocation to the factory service.

```
<soapenv:Envelope>  
...  
<soapenv:Body>  
<createResourceResponse xmlns="...">  
<wsa:EndpointReference>  
<wsa:Address>  
http://.../wsrf/services/RFTService  
</wsa:Address>  
<wsa:ReferenceProperties>  
<ns1:RFTResourceKey xmlns:ns1="...">  
8807d620-acb3-11db-9abe-b9e88f054119  
</ns1:RFTQueryResourceKey>  
</wsa:ReferenceProperties>  
...  
</wsa:EndpointReference>  
</createResourceResponse>  
</soapenv:Body>  
</soapenv:Envelope>
```

The `EndpointReference` element can be seen in this message, which contains the Address and resource key (`RFTResourceKey`) to the created service instance.

Invoking the Web Service Instance

Since the identifier of a WS-Resource instance is obtained at runtime, any message to this instance must contain the resource identifier in its SOAP header. Note that although the instance of the service is only known at runtime, the service would have been declared as a partner at development time. The BPEL specification allows for the actual service endpoint of a partner to be dynamically defined within the process. The specification, however, does not make provisions for how dynamically obtained information such as resource identifiers can be defined for those endpoints. This type of information needs to be mapped to the headers of the SOAP messages for the target

endpoint. Because the BPEL specification is deficient in this regard, the method mapping desired information to SOAP headers depends on the specific implementation of the BPEL execution engine. The method described below is suited for the ActiveBPEL Engine.

To dynamically associate an endpoint reference to a service, the WS-Addressing endpoint reference is used to represent the dynamic data required to describe a partner service endpoint. To achieve the association of a partner with its service endpoint, an endpoint reference has to be assigned to the declared partner link within the process. As shown below, we use the copy operation of an assignment activity to copy literally an endpoint reference to a variable (DynamicEndpointRef).

```
<copy>
<from>
<wsa:EndpointReference xmlns:s="...">
<wsa:Address/>
<wsa:ServiceName PortName="RFTPortType">
s:RFTService
</wsa:ServiceName>
<wsa:ReferenceProperties>
<!--Elements to be mapped to the SOAP Header-->
<wsa:Action/>
<wsa:To/>
<wsa:From/>
<ns2:RFTResourceKey/>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
</from>
<to variable="DynamicEndpointRef"/>
</copy>
```

This endpoint reference contains an Address element that will hold the service endpoint address. The ReferenceProperties of the endpoint reference contains some WS-Addressing message information header elements and a RFTResourceKey element.

The RFTResourceKey element will hold the resource identifier for the WS-Resource. Values for the endpoint reference will be assigned at run time. The message information header elements and the RFTResourceKey will be mapped by the BPEL engine to the invocation SOAP message for the partner Web service, which in this case is RFTService.

The WS-Resource identifier information required for the endpoint reference is copied from the reply message of their respective factory services. The copy operation below copies the service endpoint address from the factory response message (CreateResourceResponse) to the endpoint variable (DynamicEndpointRef).

The query attribute of the <from> and <to> clauses are XPath queries, which are used to select a field within a source or target variable part.

```
<copy>
<from variable="CreateResourceResponse"
part="response"
query="/ns4:createResourceResponse
/wsa:EndpointReference/wsa:Address"/>
<to variable="DynamicEndpointRef"
query="/wsa:EndpointReference
/wsa:ReferenceProperties/wsa:To"/>
</copy>
```

A similar mechanism is used to assign the service endpoint address to the <wsa:Address> property of the endpoint reference variable. The BPEL engine needs this address to determine the destination of the invocation message for the service. The <wsa:To> component of the message information header is used by the service to determine the endpoint of the required service instance. Here, the same address is returned by the factory because, for this application, the address of a service and its instance are the same.

```
<copy>
<from variable="CreateResourceResponse"
part="response"
query="/ns4:createResourceResponse
/wsa:EndpointReference/wsa:Address"/>
<to variable="DynamicEndpointRef"
query="/wsa:EndpointReference
/wsa:Address"/>
</copy>
```

The name of the operation to be invoked on the WS-Resource instance needs to be assigned to the Action part of the SOAP header. To achieve this, an XPath expression is used to write the name as a string to the endpoint reference variable. An XPath expression, which is specified in an expression attribute in the <from> clause, is used to indicate a value to be stored in a variable. The string that represents the operation is in the form of a URI that includes the target namespace of the WSDL document for the WS-Resource and the associated portType. Thus in the listing below, http://RFTService instance is the namespace, RFTPotType is the portType, and startTransfers is the operation.

```
<copy>
<from expression="string(
http://RFTService_instance
/RFTPotType/startTransfers)"/>
<to variable="DynamicEndpointRef"
query="/wsa:EndpointReference
/wsa:ReferenceProperties/wsa:Action" />
</copy>
```

The listing below shows how the copy operation is used and how XPath queries to copy the resource instance key (`RFTResourceKey`) from the factory response message to the endpoint reference variable.

```
<copy>
<from variable="CreateResourceResponse" part="response"
query="/ns4:createResourceResponse
/wsa:EndpointReference/wsa:ReferenceProperties
/ns2:RFTResourceKey"/>
<to variable="DynamicEndpointRef"
query="/wsa:EndpointReference/wsa:ReferenceProperties
/ns2:RFTResourceKey"/>
</copy>
```

The <wsa:From> property of the message information header identifies the source of the message. This property can be set with the WS-Addressing "anonymous" endpoint URI, which can be used because the invocation to the resource instance is synchronous in this case and the underlying message layer takes care of delivering replies to the source.

```
<copy>
<from>
```

```

<From xmlns="...">
<Address>
http://schemas.xmlsoap.org/ws/2004/03
/addressing/role/anonymous
</Address>
</From>
</from>
<to variable="DynamicEndpointRef"
query="/wsa:EndpointReference
/wsa:ReferenceProperties/wsa:From"/>
</copy>

```

After assigning values to all the necessary parts of the endpoint reference variable, an association is now made with this variable and the desired partnerlink. As shown below, a copy operation is used to copy the endpoint reference variable (`DynamicEndpointRef`) to the predefined partner link. An invocation can now be made to the Web service (WS-Resource) partner (RFT service).

The information carried in the SOAP message header of the invocation is used to identify the appropriate instance of this service.

```

<copy>
<from variable="DynamicEndpointRef"/>
<to partnerLink="rft"/>
</copy>

```

Accessing resource properties

The WS-ResourceProperties specification includes a set of port types for querying and modifying the state of a WS-Resource. The RFT service implements the `GetResourceProperty` port type of this specification. We use this port type and its operation (also called `GetResourceProperty`) to access the result from the RFT Service. Prior to invoking the `GetResourceProperty` operation, some initialization needs to be made to the variable of its input message. This initialization includes the name of the resource property to which we want to retrieve the value. In our case, this resource property is called `result`.

The listing below shows how the `GetResourcePropertyRequest` is initialized in the BPEL process. The `<from>` clause includes (as attributes) the target namespace of the WSDL documents that contain the definitions for the `GetResourceProperty` port type and the result resource property.

```

<copy>
<from>
<GetResourceProperty
xmlns="http://docs.oasis-open.org/wsrf/2004/06
/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:ns3="http://RFTService_instance">
ns3:result
</GetResourceProperty>
</from>
<to variable="GetResourcePropertyRequest"
part="GetResourcePropertyRequest"/>
</copy>

```

The WS-ResourceProperties specification also includes the `GetMultipleResourceProperty` port type for retrieving the values of multiple resource properties. To invoke the operation of this port type, the variable message initialization must include the list of all the resources properties, each encapsulated within a `</ResourceProperty>` element.

```
<copy>
<from>
<GetMultipleResourceProperty
  xmlns="http://docs.oasis-open.org/wsrf/2004/06
  /wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
  <ResourceProperty
    xmlns:ns3="http://RFTService_instance">
    ns3:result
  </ResourceProperty>
  <ResourceProperty xmlns:ns1="...">
    ...
  </ResourceProperty>
</GetMultipleResourceProperty>
</from>
<to .../>
</copy>
```

Because an attempt is being made to try and access the resource properties of a WS-Resource instance, assignments need to be made to all parts of the message header necessary for identifying the instance. The way to do this is previously described. The only difference in this example is in the URI that specifies the verb of the invocation message.

```
<copy>
<from expression="string('http://docs.oasis-open.org
  /wsrf/2004/06/wsrf-WS-ResourceProperties
  /GetResourceProperty')"/>
<to variable="DynamicEndpointRef"
  query="/wsa:EndpointReference
  /wsa:ReferenceProperties/wsa:Action"/>
</copy>
```

API Sample for WorkflowFactory and WorkflowImpl Services:

```
// create a new workflow from a bpel file
WorkflowFactoryServiceClient factoryClient = new WorkflowFactoryServiceClient(
    "https://cagrid-workflow.nci.nih.gov:8443/wsrf/services/cagrid/WorkflowFactoryService"
);
WMSInputType input = createInput(null, "Test1.bpel");
WMSOutputType wmsOutput = factoryClient.createWorkflow(input);
EndpointReferenceType epr = wmsOutput.getWorkflowEPR();
WorkflowServiceImplClient serviceClient = new WorkflowServiceImplClient(epr);

// load workflow input from an xml file
StartInputType startInput = new StartInputType();
WorkflowInputType inputArgs = new WorkflowInputType();
FileInputStream in = new FileInputStream("input.xml");
Element inputEl = XmlUtils.newDocument(in).getDocumentElement();
MessageElement anyContent = AnyHelper.toAny(new MessageElement(inputEl));
inputArgs.set_any(new MessageElement[] {anyContent});
startInput.setInputArgs(inputArgs);

// start the workflow
WorkflowStatusType status = serviceClient.start(startInput);

// sleep while active
Object sleep = new Object();
do {
    try { sleep.wait(1000); } catch (InterruptedException e) { e.printStackTrace(); }
    status = serviceClient.getStatus();
} while (WorkflowStatusType._Active.equals(status.getValue()));

// get output
WorkflowOutputType output = serviceClient.getWorkflowOutput();
String outputXml = AnyHelper.toSingleString(output.get_any());
```


Chapter 9 caGrid Global Model Exchange

This chapter describes the caGrid Global Model Exchange.

Topics in this chapter include:

- Overview on this page
- GME Client on page 191

Overview

The caGrid Global Model Exchange (GME) is dependent on several software packages/systems that must be installed prior to installing and deploying the GME. The GME requires all of the required software packages of the caGrid core (Table 9-1).

Software	Version	Description
Java SDK	jsdk1.5 or higher	GME is written in Java therefore it requires the Java SDK. After installing you will have to set up an environmental variable pointing to the Java SDK directory and name it JAVA_HOME.
Mysql	Mysql 4.x or higher	For persistence and cache of models, GME uses the mysql database. Mysql can be downloaded from http://www.mysql.com/products/mysql/ . The GME requires mysql version 4.X.
Ant	Ant 1.6.5	GME along with the Globus Toolkit in which GME is built on, uses Jakarta Ant for building and deploying.
Globus	Globus 4.0.3	GME is built on top of the Globus Toolkit. GME requires the ws-core installation of the Globus Toolkit.
Tomcat (Only required if deploying to Tomcat)	Tomcat 5.0.30	GME can be optionally deployed as a Grid Service to a Tomcat deployed Globus Toolkit.

Table 9-1 Software prerequisites for GME

Building the GME

GME is built when caGrid core is built. To build caGrid, the following environment variables are required:

- `GLOBUS_LOCATION` - The location of your globus 4.0.X installation.
- `CATALINA_HOME` - The location of your Tomcat installation. (optional container)

If you have checked out caGrid core from CVS, installed the required software packages, and set the required environment variables, begin building caGrid core by going into the caGrid core checkout directory and entering `ant all`.

Configuring and Deploying the GME

Configuration

GME requires two configuration files for configuring the service. One configuration file is for configuring the Mobius GME and the other is for configuring the GME security preferences.

Each addresses a different GME set-up: connecting GME to other services, etc. The file `gme-globus-config.xml` contains a basic setup for a GME server to run on the local machine and provides an example of how the configuration files are structured. See <http://projectmobius.org/docs/mobiusconfig.php> for information on the elements in the top resource block of the configuration files.

The `localhost config` files should never be used for more than single GME testing purposes. If you are running multiple GMEs, customize the config files by assigning unique service identifiers and making any other changes necessary to specify the service as unique. When other services connect to a GME started as localhost, the GME identifies itself as "localhost" to the connecting service. This can cause problems with service name resolution.

The GME server contains a single resource block that provides certain functionality and information to other GME components. The resources defined in the block are instantiated by the GME server at startup and are configured by the config file. The resource for the GME is: GME Configuration `<resource name="gmeConfig" . . . >`. Each configuration element is listed here, along with its children. Below this list, the purpose of the elements are described.

- *policies*
- *performance-caching*
- *notification-policy*
- *root-database*

<policies>

This element establishes parameters for how long the GME should keep old data and what other hosts it should notify of its existence.

<performance-caching>

The `<namespace-caching>` and `<schema-caching>` sub-elements of this element tell the GME how much namespace and schema data it should cache and for how long it should maintain the cache.

<notification-policy>

This element contains a `<notification-list>` sub-element that specifies which running GMEs it should notify upon instantiation.

<root-database>

This element configures the MySQL root database information. The children of this element configure the database. Its "id" attribute is the base name of the MySQL

databases that will be created and used by the GME. If you configure multiple GMEs to use a single MySQL installation, be sure to give a unique value to the "id" for each GME or there will be problems with name collision in MySQL.

<name>

This element specifies the name of the database. For MySQL's root database, this should be nothing.

<driver>

The driver class for accessing the MySQL database.

<urlPrefix>

The URL prefix for accessing the database.

<host>

The host the database lives on. Usually, this will be localhost.

<port>

The port from which the database can be accessed.

<username>

The username to log into the database with. This username must have privileges to create and delete databases and tables.

<password>

The password to authenticate the username.

<pool>

This element determines how many connections to the database will be made initially. When the GME needs to communicate with the database, it will get a connection and when it's done, it releases it. Should the Database Manager run out of available connections, it will make a new one, but this causes a slight delay. Set the pool value in anticipation of how many concurrent database operations will be needed.

Deployment

The `GME_LOCATION/deploy.properties` file allows the configuration of deployment time properties of the service. The properties file contains two variables for configuring the service name and the service path, as well as information needed to identify the service. These variables are defaulted during skeleton creation time.

```
service.name=GlobalModelExchange
service.deployment.path=cagrid/GlobalModelExchange
service.deployment.host.default=localhost
service.deployment.port.default=8080
service.deployment.protocol.default=http
```

Once GME is configured, it can be deployed to Globus running in Tomcat by entering `ant deployTomcat` or `ant deployGlobus` from the `GME_LOCATION` directory. These targets will prompt for the *host*, *port*, and *protocol* which the GME will be running with, and can be changed from the defaults set above in the `deploy.properties`.

Note: For proper functioning of the GME, these values must be set according to the deployment environment in use.

Once deployed starting or restarting the container starts up the GME.

GME Backup and Restore

GME installations should run a backup script to make sure the integrity of the database can be restored if there are any failures. A general purpose script for this is provided in the `tools/backup` directory. There is one script for backup and one for restore. Each script has a short description inside describing the usage of the script and what variables might need to be configured. This script can be executed from a crontab and maintains five rolling backup caches of the GME databases.

GME Move

If the GME is started up with a different service URL or a move if required to another machine, the GME database will need to be updated. Scripts are provided in the `tools/move` directory for exporting and importing a GME database. Scripts are also provided for updating the service URL in the database if the service has a new URL resulting from either moving or changing the address of the machine etc. In order to move a GME, the following steps are required.

1. Export the database:
 - a. Use the `gmeExportDB.sh` script to create an archive of the database data
 - `./gmeExportDB.sh [databaseprefix] [optional database password]`
2. Import the database:
 - a. First make sure that you have started up the container on the new machine with the gme service deployed to it. This will create the database and its required tables.
 - b. Use the `gmelImportDB.sh` script to import the archived data created from the export step.
 - `./gmelImportDB.sh [databaseprefix] [data file] [optional database password]`
3. Update the service URL in the database:
 - a. Use the `gmeChangeURL.sh` script to modify the databases service url entries with the new service URL.
 - `./gmeChageURL.sh [databaseprefix] [old service url] [new service url] [optional database password]`

Important Notes

The default GME configuration is set to connect to a MySQL database on the localhost with no password and username root. If you need to change this be sure to edit the `gme`

config file in the etc directory. The GME also dynamically creates its databases and tables. Make sure that the database privileges are set correctly to enable this.

GME Client

GME Client API

The GME client API is a simple java based API that enables simple access to remote GMEs in order to publish, retrieve, and discover models as well as manage GMEs. The client API comes from the Mobius project and is included in the caGrid release. The communication factories required to use the GME in the caGrid environment are built into the caGrid release jar and deployed into the skeleton. Below is a code example of how to get a handle to a GME and request a schema, and all its referenced schemas, to be retrieved and written to a place on the file system.

```
GridServiceResolver.getInstance().setDefaultFactory(new
GlobusGMEXMLDataModelServiceFactory());
List writtenNamespaces = null;
File directory = new File(CACHE_LOCATION);
try {
XMLDataModelService handle = (XMLDataModelService)
GridServiceResolver.getInstance()
.getGridService("http://dc04.bmi.ohio-
state.edu:8080/ogsa/services/cagrid/gme");
writtenNamespaces = handle.cacheSchema(cagrid.nci.nih.gov/1/Gene,directory);
} catch (MobiusException e1) {
e1.printStackTrace();
}
```

For more information on other methods the GME API provides, refer to <http://projectmobius.org/docs/gmeapi.php> or browse the Mobius GME source code, which is freely available at www.projectmobius.org.

GME Viewer

To launch the GME Viewer, run *ant gmeViewer* from the GME_LOCATION. This launches the Mobius GME Viewer GUI configured to use Globus for communication. Once this tool is launched, follow the Mobius GUI Documentation for using the GUI (<http://projectmobius.org/docs/gmeqs.php>).

A simplified GUI for communicating with the GME is also provided as an Introduce extension, under the “Browse Data Types” menu.

Appendix A References

Scientific Publications

-
- [1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and T. S., "Data Management and Transfer in High Performance Computational Grid Environments," *Parallel Computing Journal*, vol. 28, pp. 749-771, 2002.
 - [2] W. E. Allcock, I. Foster, and R. Madduri, "Reliable Data Transport: A Critical Service for the Grid.,," in *Proceedings of Building Service Based Grids Workshop, Global Grid Forum 11*. Honolulu, Hawaii, USA, 2004.
 - [3] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Cactus-G Toolkit: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments," in *Proceedings of the 4th Globus Retreat*. Pittsburg, PA, 2000.
 - [4] H. Andrade, T. Kurc, A. Sussman, and J. Saltz, "Active Proxy-G: Optimizing the Query Execution Process in the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.
 - [5] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.
 - [6] M. P. Atkinson and et.al., "Grid Database Access and Integration: Requirements and Functionalities," Technical Document, Global Grid Forum. <http://www.cs.man.ac.uk/grid-db/documents.html>, 2002.
 - [7] F. Berman, H. Casanova, J. Dongarra, I. Foster, C. Kesselman, J. Saltz, and R. Wolski, "Retooling Middleware for Grid Computing," *NPACI & SDSC enVision*, vol. 18, 2002.
 - [8] M. Beynon, T. Kurc, A. Sussman, and J. Saltz, "Design of a Framework for Data-Intensive Wide-Area Applications," in *Proceedings of the 2000 Heterogeneous Computing Workshop (HCW2000)*. Cancun, Mexico, 2000.
 - [9] H. Casanova, O. Graziano, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2000)*: ACM Press/IEEE Computer Society Press, 2000.
 - [10] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*: ACM Press/IEEE Computer Computer Society Press, 2002, pp. 1-17.
 - [11] A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshahi, B. Drach, D. Williams, and D. Middleton, "High-performance remote access to climate simulation data: a challenge problem for data grid technologies," *Parallel Computing*, vol. 29, pp. 1335-1356, 2003.
 - [12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, pp. 187-200, 2000.

- [13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, pp. 25-39, 2003.
- [14] E. Deelman, G. Singh, M. P. Atkinson, A. Chervenak, N. P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su, "Grid-Based Metadata Services," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, 2004.
- [15] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit..," *International Journal of High Performance Computing Applications*, vol. 11, pp. 115--128, 1997.
- [16] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *Proceedings of the 14th Conference on Scientific and Statistical Database Management (SSDBM '02)*, 2002.
- [17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computational Management Agent for Multi-institutional Grids," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*: IEEE Press, 2001.
- [18] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: An Open Grid Service Architecture Implemented with JINI," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.
- [19] A. S. Grimshaw and W. Wulf, "The Legion: Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40, pp. 39--45, 1997.
- [20] S. Hastings, S. Langella, S. Oster, and J. Saltz, "Distributed Data Management and Integration: The Mobius Project," *Proceedings of the Global Grid Forum 11 (GGF11 Semantic Grid Applications Workshop, Honolulu, Hawaii, USA.)*, pp. 20-38, 2004.
- [21] S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, and J. Saltz, "Dorian: Grid Service Infrastructure for Identity Management and Federation," presented at The 19th IEEE Symposium on Computer-Based Medical Systems, Special Track: Grids for Biomedical Informatics, Salt Lake City, Utah., 2006.
- [22] R. Oldfield and D. Kotz, "Armada: A Parallel File System for Computational Grid," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid2001)*. Brisbane, Australia: IEEE Computer Society Press, 2001.
- [23] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi, "Ninf: A Network based Information Library for a Global World-Wide Computing Infrastructure," in *Proceedings of the Conference on High Performance Computing and Networking (HPCN '97) (LNCS-1225)*, 1997, pp. 491-502.
- [24] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman, "A Metadata Catalog Service for Data Intensive Applications," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2003)*, 2003.
- [25] G. Singh, E. Deelman, G. Mehta, K. Vahi, M. Su, B. Berriman, J. Good, J. Jacob, D. Katz, A. Lazzarini, K. Blackburn, and S. Koranda, "The Pegasus Portal: Web Based Grid Computing," in *Proceedings of the 20th Annual ACM Symposium on Applied Computing*. Santa Fe, New Mexico, 2005.
- [26] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. Fernandes, and R. Sakellariou, "Distributed Query Processing on the Grid.,," presented at Proceedings of the Third Workshop on Grid Computing (GRID2002), Baltimore, MD, 2003.
- [27] D. Thain, J. Basney, S. Son, and M. Livny, "Kangaroo Approach to Data Movement on the Grid," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC-10)*, 2001.

- [28] L. Weng, G. Agrawal, U. Catalyurek, T. Kurc, S. Narayanan, and J. Saltz, "An Approach for Automatic Data Virtualization," in *Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*. Honolulu, Hawaii, 2004, pp. 24-33.
- [29] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure Working Group Technical Report, Global Grid Forum.
<http://www.globus.org/alliance/publications/papers/ogsa.pdf> 2002.
- [30] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations.,," *International Journal of Supercomputer Applications*, vol. 15, pp. 200-222, 2001.
- [31] E. Cerami, *Web Services Essentials*: O'Reilly & Associates Inc., 2002.
- [32] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*: SAMS Publishing, 2002.
- [33] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The WS-Resource Framework version 1.0," vol. 2004, 2004.
- [34] J. Saltz, S. Oster, S. Hastings, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: Design and Implementation of the Core Architecture of the Cancer Biomedical Informatics Grid," *Bioinformatics. (in press)*. 2006.
- [35] S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, and J. Saltz, "A Distributed Data Management Middleware for Data-Driven Application Systems," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster 2004)*, 2004.
- [36] K. Bhatia, S. Chandra, and K. Mueller, "GAMA: Grid Account Management Architecture," San Diego Supercomputer Center (SDSC), UCSD Technical Report. #TR-2005-3, 2005.
- [37] I. Foster, C. Kesselman, S. Tuecke, V. Volmer, V. Welch, R. Butler, and D. Engert, "A National Scale Authentication Infrastructure," *IEEE Computer*, vol. 33, pp. 60-66, 2000.
- [38] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," presented at 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
- [39] H. Morohoshi and R. Huang, "A User-friendly Platform for Developing Grid Services over Globus Toolkit 3," presented at The 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05), 2005.
- [40] S. Mizuta and R. Huang, "Automation of Grid Service Code Generation with AndroMDA for GT3," presented at The 19th International Conference on Advanced Information Networking and Applications (AINA'05), 2005.
- [41] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, pp. 643-662, 2001.
- [42] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke, "CoG Kits: A Bridge Between Commodity Distributed Computing and High Performance Grids," presented at ACM Java Grande 2000 Conference, 2000.
- [43] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report," presented at the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004), New Jersey, USA, 2004.
- [44] M. Humphrey and G. Wasson, "Architectural Foundations of WSRF.NET," *International Journal of Web Services Research*, vol. 2, pp. 83-97, 2005.
- [45] M. Smith, T. Friese, and B. Freisleben, "Model Driven Development of Service Oriented Grid Applications," presented at Advanced International Conference on

Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW '06), 2006.

Technical Manuals/Articles

National Cancer Institute. "caCORE 3.1 Technical Guide",
ftp://ftp1.nci.nih.gov/pub/cacore/caCORE3.1_Tech_Guide.pdf

Java Bean Specification: <http://java.sun.com/products/javabeans/docs/spec.html>

Foundations of Object-Relational Mapping:
<http://www.chimu.com/publications/objectRelational/>

Object-Relational Mapping articles and products:

<http://www.service-architecture.com/object-relational-mapping/>

Hibernate Reference Documentation:
http://www.hibernate.org/hib_docs/reference/en/html/

Basic O/R Mapping: http://www.hibernate.org/hib_docs/reference/en/html/mapping.html

Java Programming: <http://java.sun.com/learning/new2java/index.html>

Javadoc tool: <http://java.sun.com/j2se/javadoc/>

JUnit: <http://junit.sourceforge.net/>

Extensible Markup Language: <http://www.w3.org/TR/REC-xml/>

XML Metadata Interchange: <http://www.omg.org/technology/documents/formal/xmi.htm>

Global Grid Forum: <http://www.gridforum.org>

Globus: <http://www.globus.org>

Mobius: <http://www.projectmobius.org>

W3C: <http://www.w3c.org>

OGSA-DAI: <http://www.ogsadai.org>

Apache: <http://www.apache.org>

Globus Toolkit 3 Programmer's Tutorial:

http://gdp.globus.org/gt3-tutorial/singlehtml/progtutorial_0.4.3.html

XPath tutorial: http://www.w3schools.com/xpath>xpath_syntax.asp

Globus Security Overview:

<http://www.ogsadai.org.uk/docs/OtherDocs/SECURITY-FOR-DUMMIES.pdf>

High level Overview of Grid:

<http://gridcafe.web.cern.ch/gridcafe/index.html>

Overview of Globus Toolkit 3 and the OGSI architecture :

<http://www-128.ibm.com/developerworks/grid/library/gr-gt3/>

caBIG Material

caBIG: <http://cabig.nci.nih.gov/>

caBIG Compatibility Guidelines: http://cabig.nci.nih.gov/guidelines_documentation

caCORE Material

caCORE: <http://ncicb.nci.nih.gov/NCICB/infrastructure>

caBIO: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO

caDSR: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr

EVS: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary

CSM: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm

Appendix B Glossary

Term	Definition
API	Application Programming Interface
Authz	caGrid Authorization component
BPEL	Business Process Execution Language
CA	Certificate Authority
caArray	cancer Array Informatics
caBIG	cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caCORE	cancer Common Ontologic Representation Environment
caDSR	Cancer Data Standards Repository
caGrid	Current test bed architecture of caBIG
CRL	Certificate Revocation List
CSM	Common Security Module
CVS	Concurrent Versions System
DAO	Data Access Objects
DN	Distinguished Name
IdP	Identity Provider
EPR	End Point Reference
EVS	Enterprise Vocabulary Services
GAARDS	Grid Authentication and Authorization with Reliably Distributed Services
GDE	Introduce Graphical Development Environment
GForge	Primary site for collaborative project development for the NCI Center for Bioinformatics (NCICB) and for the NCI's Cancer Biomedical Informatics Grid™ (caBIG)
GGF	Global Grid Forum
GME	Mobius Global Model Exchange - DNS-like service for the universal creation, versioning, and sharing of data descriptions
Grid Service	Basically a Web Services with improved characteristics and standard services like stateful and potentially transient services, Service Data, Notifications, Service Groups, portType extension, and Lifecycle management.
GSH	Grid Service Handle
GSI	Grid Security Infrastructure - represents the latest evolution of the Grid Security Infrastructure. GSI in GT3 builds off of the functionality present in early GT2 toolkit releases - X.509 certificates, TLS/SSL for authentication and message protection, X.509 Proxy Certificates for delegation and single sign-on.

Term	Definition
GTS	Grid Trust Service - maintains a federated trust fabric of all the trusted digital signers in the grid
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JDBC	Java Database Connectivity
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
LDAP	Lightweight Directory Access Protocol
MAGE	MicroArray and Gene Expression
MAGE-OM	MicroArray Gene Expression - Object Model
Metadata	Definitional data that provides information about or documentation of other data.
MGED	Microarray Gene Expression Data
Mobius	An array of tools and middleware components to coherently share and manage data and metadata in a Grid and/or distributed computing environment.
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
OGSA	Open Grid Services Architecture - developed by the Global Grid Forum, aims to define a common, standard, and open architecture for grid-based applications.
OGSI	Open Grid Services Infrastructure -gives a formal and technical specification of what a Grid Service is. In other words, for a high-level architectural view of what Grid Services are, and how they fit into the next generation of grid applications
PKI	Public Key Cryptography
RDBMS	Relational Database Management System
SAML	Secure Access Markup Language
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TRA	Trusted Registration Authority
UI	User Interface
UID	User Identification
UML	Unified Modeling Language

Term	Definition
UPT	User Provisioning Tool
URL	Uniform Resource Locators
Virtualization	Make a computational or data resource available to caBIG community - some people call "Gridification"
VO	Virtual Organization
WAR	Web Application Archive
Web Service	Application to application communication using web based service interfaces as describe by the Web Services 1.0 or 2.0 specification.
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
X.509 Certificate	With its corresponding private key forms a unique credential or so-called "grid credential" within the grid
XMI	XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments
XML	Extensible Markup Language (http://www.w3.org/TR/REC-xml/) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML
XPath	XML query/traversal language adhering to the XPath specification set forth by the W3C.

Index

addNonNullPredicate
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 66

annotateServiceMetadata, 74

API
caDSR grid service usage, 41, 73
caDSR Grid Service usage examples, 80
caDSR Grid Services details, 76
Discovery API details, 52
Discovery usage examples, 69
Discovery usage overview, 50
EVS usage overview, 84
federated query processor details, 161
federated query processor examples, 166
GME client, 199
metadata details, 44
metadata usage examples, 46

BPEL, 181

buildDataUMLClassPredicate
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 66

buildPOCPredicate
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 65

buildUMLClassPredicate
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 65

caBIG
references, 205

caCORE
references, 205

caDSR Grid Service
overview, 41, 73
security considerations, 76

caDSR Grid Services
API details, 76
API usage examples, 80
examining project information model, 80
generating data service metadata, 81

caDSRServiceClient, 76

caGrid
security overview, 91

createConceptPredicatedUMLClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 61

createPermissibleValuePredicatedUMLClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 60

DataServiceConstants, 142

deserializeDomainModel, 49
gov::nih::nci::cagrid::metadata::MetadataUtils, 46

deserializeServiceMetadata, 49
gov::nih::nci::cagrid::metadata::MetadataUtils, 45

discoverByFilter

gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 66

discoverDataServicesByAssociationsWithClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 64

discoverDataServicesByDomainModel, 71
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 62

discoverDataServicesByExposedClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 63

discoverDataServicesByModelConceptCode
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 62

discoverDataServicesByPermissibleValue
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 64

discoverServiceByOperationInput, 72

discoverServicesByConceptCode, 71
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 56

discoverServicesByDataConceptCode
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 59

discoverServicesByName
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 55

discoverServicesByOperationClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 58

discoverServicesByOperationConceptCode, 71
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 59

discoverServicesByOperationInput
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 57

discoverServicesByOperationName, 71
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 56

discoverServicesByOperationOutput
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 57

discoverServicesByPermissibleValue
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 60

discoverServicesByPointOfContact
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 54

discoverServicesByResearchCenter, 71
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 54

discoverServicesBySearchString, 70
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 53

Discovery API
configuring index service, 69
details, 52

discovering services, 70
 usage examples, 69

DiscoveryClient
 definition, 50
`gov.nih.nci.cagrid.discovery.client:DiscoveryClient`, 52

Discovery API
 overview, 50

Document text conventions, 2

EVS API
`getHistoryRecords`, 87
`getMetaSources`, 84
`getVocabularyNames`, 85
`searchDescLogicConcept`, 85
`searchMetaThesaurus`, 88
`searchSourceByCode`, 89
 usage overview, 84

execute
`gov.nih.nci.cagrid.fqp.processor:FederatedQueryEngine`, 164

executeAndAggregateResults
`gov.nih.nci.cagrid.fqp.processor:FederatedQueryEngine`, 165

FederatedQueryEngine
`gov.nih.nci.cagrid.fqp.processor:FederatedQueryEngine`, 164

FileReader, 49

FileWriter, 48

generateDomainModelForClasses, 74

generateDomainModelForClassesWithExcludes, 74

generateDomainModelForPackage, 73

generateDomainModelForProject, 73, 82

getAllDataServices
`gov.nih.nci.cagrid.discovery.client:DiscoveryClient`, 61

getAllServices
`gov.nih.nci.cagrid.discovery.client:DiscoveryClient`, 53

getDomainModel
`gov.nih.nci.cagrid.metadata.MetadataUtils`, 45

getIndexEPR
`gov.nih.nci.cagrid.discovery.client:DiscoveryClient`, 67

getServiceMetadata
`gov.nih.nci.cagrid.metadata.MetadataUtils`, 44

Global Model Exchange. See GME

GME
 building, 195
 client API, 199
 configuring, 195
 deploying, 197
 software prerequisites, 195
 viewer, 199

GME XE "Global Model Exchange" \t "See
GME" XE "GME:software prerequisites"
 overview, 195

gov.nih.nci.cagrid.discovery.client:DiscoveryClient
`addNonNullPredicate`, 66
`buildDataUMLClassPredicate`, 66
`buildPOCPredicate`, 65
`buildUMLClassPredicate`, 65
`createConceptPredicatedUMLClass`, 61
`createPermissibleValuePredicatedUMLClass`, 60
`discoverByFilter`, 66
`discoverDataServicesByAssociationsWithClass`, 64
`discoverDataServicesByDomainModel`, 62
`discoverDataServicesByExposedClass`, 63
`discoverDataServicesByModelConceptCode`, 62
`discoverDataServicesByPermissibleValue`, 64
`discoverServicesByConceptCode`, 56
`discoverServicesByDataConceptCode`, 59
`discoverServicesByName`, 55
`discoverServicesByOperationClass`, 58
`discoverServicesByOperationConceptCode`, 59
`discoverServicesByOperationInput`, 57
`discoverServicesByOperationName`, 56
`discoverServicesByOperationOutput`, 57
`discoverServicesByPermissibleValue`, 60
`discoverServicesByPointOfContact`, 54
`discoverServicesByResearchCenter`, 54
`discoverServicesBySearchString`, 53
`DiscoveryClient`, 52
`getAllDataServices`, 61
`getAllServices`, 53
`getIndexEPR`, 67
`main`, 68
`setIndexEPR`, 68
`translateXPath`, 67

gov.nih.nci.cagrid.discovery.XPathUtils
`translateXPath`, 68

gov.nih.nci.cagrid.fqp.processor:FederatedQueryEngine
`execute`, 164
`executeAndAggregateResults`, 165
`FederatedQueryEngine`, 164

gov.nih.nci.cagrid.metadata.MetadataUtils
`s`
`deserializeDomainModel`, 46
`deserializeServiceMetadata`, 45
`getDomainModel`, 45
`getServiceMetadata`, 44
`serializeDomainModel`, 46
`serializeServiceMetadata`, 45

Grid Authentication and Authorization with Reliably Distributed Services. See GAARDS

Grid Grouper
 overview, 94

InternalRuntimeException, 44, 51

InvalidProjectException, 73

InvalidResourcePropertyException, 44, 51

main
`gov.nih.nci.cagrid.discovery.client:DiscoveryClient`, 68

Metadata
 accessing from service, 46

API details, 44
API usage examples, 46
processing as XML, 48
PointOfContact, 72
QName, 143
QueryInvalidOperationException, 44, 51
Reader, 49
References
 caBIG, 205
 caBIG materials, 205
 caCORE, 205
 caCORE material, 205
 scientific publications, 201
 technical manuals, guides, 204
RemoteResourcePropertyRetrievalException,
 44, 51
ResourcePropertyHelper, 43
ResourcePropertyRetrievalException, 44, 50, 51
serializeDomainModel, 48
 gov::nih::nci::cagrid::metadata::MetadataUtils,
 46
serializeServiceMetadata, 48
 gov::nih::nci::cagrid::metadata::MetadataUtils, 45
ServiceMetadata, 49, 74
setIndexEPR
 gov::nih::nci::cagrid::discovery::client::DiscoveryCli
 ent, 68

sourceClassName, 74
sourceRoleName, 74
targetClassName, 74
targetRoleName, 74
thread, 143
translateXPath
 gov::nih::nci::cagrid::discovery::client::DiscoveryCli
 ent, 67
 gov::nih::nci::cagrid::discovery::XPathUtils, 68
UMLAssociation, 76
UMLAssociationExclude, 75
UMLAssociationExcludes, 74
UMLAssociationMetadata, 76
UMLClass, 72
URL, 142
Workflow
 architecture description, 179
 overview, 179
 security contexts, 187
 WorkflowFactoryService API, 181
 WorkflowManagementService API, 183
Writer, 48
WS-Enumeration
 client API, 172
 overview, 172