****

Zentity (Version 2.0)

Quick Guide to Zentity Core Features

May 2009

Contents

[Introduction 3](#_Toc229469431)

[Core Features 3](#_Toc229469432)

[Basic features & usage 4](#_Toc229469433)

[Default UI 4](#_Toc229469434)

[Home 4](#_Toc229469435)

[Browse 5](#_Toc229469436)

[Manage 6](#_Toc229469437)

[Services 7](#_Toc229469438)

[RSS/ ATOM Feed 7](#_Toc229469439)

[AtomPub 8](#_Toc229469440)

[SWORD 11](#_Toc229469441)

[Platform API 13](#_Toc229469442)

[Search 13](#_Toc229469443)

[Security 16](#_Toc229469444)

[Data Models 24](#_Toc229469445)

[Core data model 24](#_Toc229469446)

[Scholarly works data model 27](#_Toc229469447)

[More Features 29](#_Toc229469448)

# Introduction

[Zentity](http://research.microsoft.com/zentity) (Version 2.0) aims to provide the necessary building blocks, tools, and services for creating and managing an organization’s repository ecosystem around research content.

This platform is built on top of Microsoft SQL Server and the ADO.Net Entity Framework (part of the Microsoft .NET Framework 4.0) which provides a [LINQ](http://msdn.microsoft.com/en-us/data/cc299380.aspx) experience for developers building on top of the platform.

## Core Features

High level repository features include:

* **Core data model** with extensibility, which can be used to create custom data models, even for domains other than Scholarly Communications
* **Built-in Scholarly Works data model** with predefined resources
* **Extensive Search** similar to Advanced Query Syntax ([AQS](http://msdn.microsoft.com/en-us/library/aa965711(VS.85).aspx))
* **Pluggable Authentication** and Authorization Security API
* **Basic Web-based User Interface** to browse and manage resources with reusable custom controls
* **RSS/ATOM, OAI-PMH, AtomPub, SWORD Services** for exposing resource information
* **Extensive help** with code samples extend the platform by developers

Zentity provides an ecosystem of tools and services to manage resources, their metadata, and relationships between them. Various built-in pluggable components provide tools to manage the resources.

# Basic features & usage

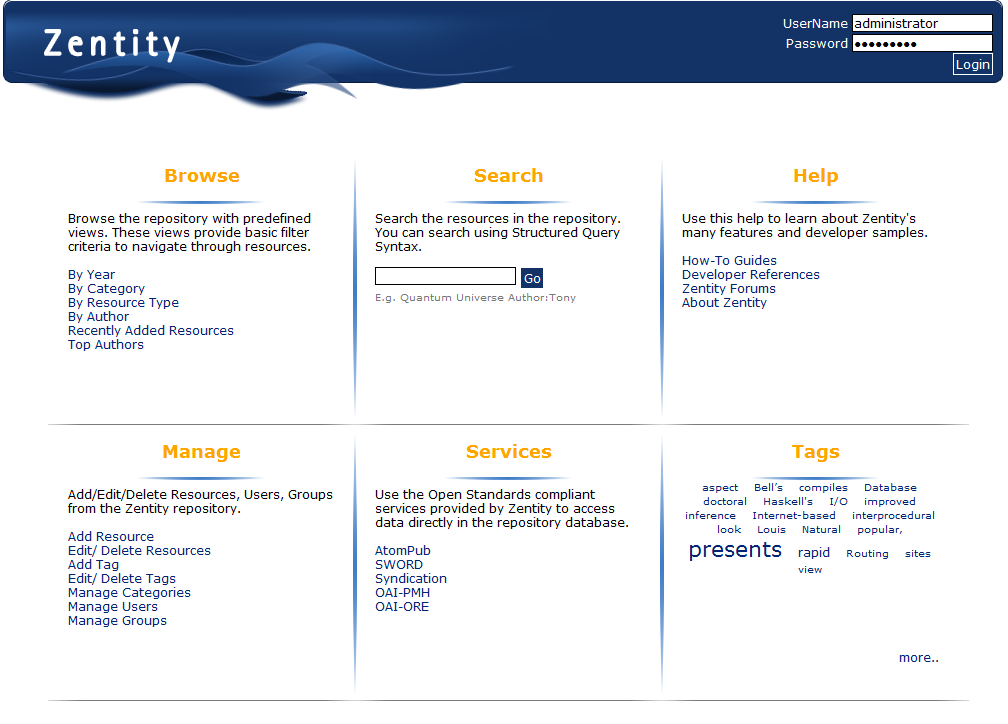
## Default UI

Zentity provides an out of the box default UI which shows usage of all components put together.

* **Manage** resources, users, permissions
* **Browse** using predefined views
* **Search** using logical query string
* **Service** endpoints
* **Help** to learn more about this repository platform

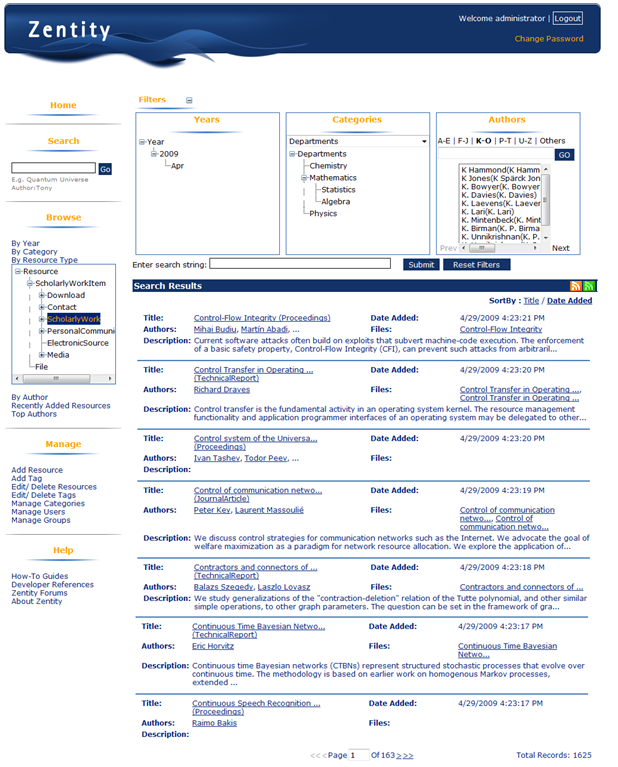
### Home

The home page provides an overview of the various features available as part of Zentity. Each section depicts a functionality set.



### Browse

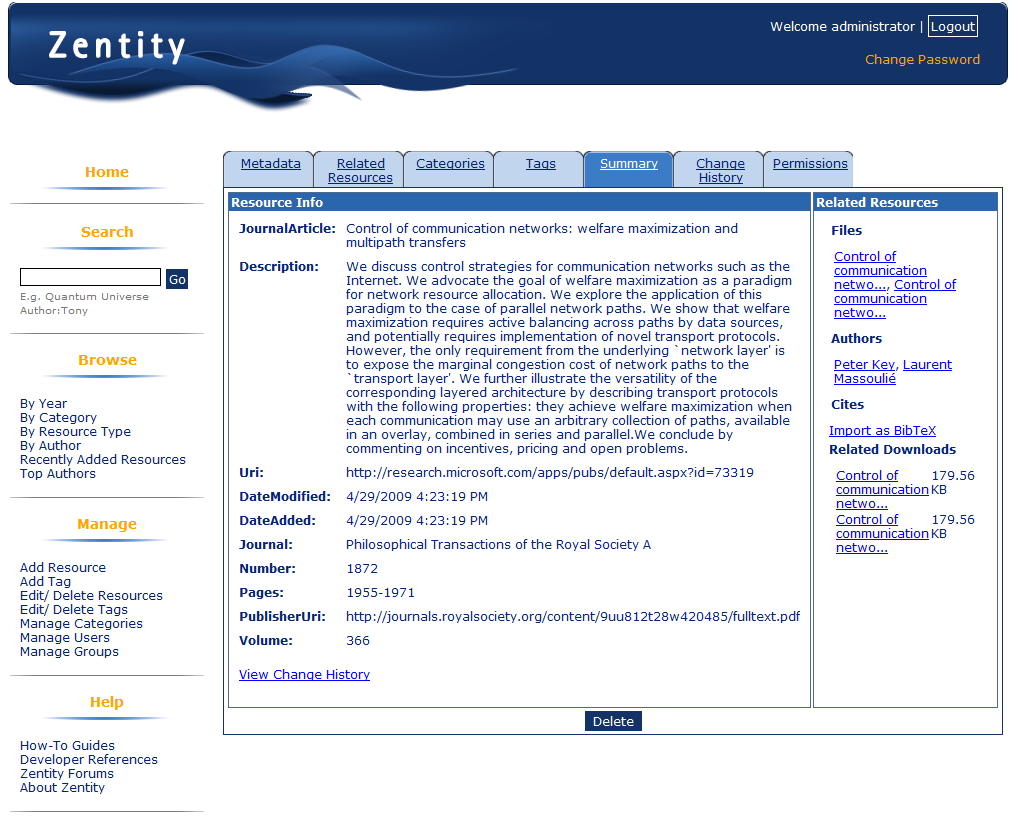
Browsable views allow users to explore resources based on basic filters provided on left navigation bar. Users can then narrow down results by adding more filters through the top filter pane on right.



### Manage

Manage pages allow a user to add/edit/delete resources. The tabbed interface allows a user to manage metadata, associations with other related resources, tags and categories.

An owner can also track the changes to resources via change history and also can manage permissions.



## Services

Zentity provides a set of services to that can be used present resource information. These services include syndication (RSS/ ATOM), SWORD.

### RSS/ ATOM Feed

Zentity provides both RSS & ATOM Syndication support for the resources in the database, based on the search queries. For more detail go to following link on your installed zentity server:

http://<webserver>:<portnumber>/Syndication/Syndication.ashx/Help

A web browser can be used as a client for the syndication service. Syndication does not support full text search queries, hence the feeds will be available for non content search queries only.

e.g. – if a user wants to create a Atom feed for all the Books created by “Foo Bar”, then he can use following URL to get the required feed:

|  |
| --- |
| http://webserver:portnumber/Syndication/Syndication.ashx?resourcetype: Book author:(firstName:Foo lastName:Bar) |

The XML response would be formatted like,

|  |
| --- |
| <feed xml:lang="en-US" xmlns="http://www.w3.org/2005/Atom">  <title type="text">Zentity feed for search query : resourcetype: Book author:(firstName:Foo lastName:Bar)</title>  <subtitle type="text">Zentity feed for provided search query</subtitle>  <id>uuid:6568981d-abaa-45ef-b8be-e36736a59d3a;id=1</id>  <rights type="text">@Copyrights</rights>  <updated>2009-04-28T14:52:26+05:30</updated>  <author>  <name>Your Name</name>  <email>email@domain.com</email>  </author>  <generator>Zentity Feed Services</generator>  <link rel="alternate" href="http://webserver:portnumber/Syndication/Syndication.ashx?resourcetype:%20Book%20author:(firstName:Foo%20lastName:Bar)" />  <entry>  <id>urn:guid:dec98f3a-443b-42e9-9ea0-f74abbe8e98c</id>  <title type="text">My Book</title>  <updated>2009-04-28T14:47:28+05:30</updated>  <author>  <name>Foo Bar</name>  </author>  <link rel="edit" href="http://MyServer:9090/AtomPub/AtomPub.ashx/Book/edit/dec98f3a-443b-42e9-9ea0-f74abbe8e98c" />  <content type="text" />  </entry>  </feed> |

### AtomPub

The Zentity AtomPub (Atom Publishing Protocol) service can be used for deposit or edit resources into Zentity repository. For more information on AtomPub please refer [Atom Publishing Protocol](http://www.ietf.org/rfc/rfc5023.txt)

#### Service Document

The service document provides all supported collection URIs in a workspace and the accepted media file types for each collection. The service Document URI will be in the form of <http://MyServer:9090/AtomPub/AtomPub.ashx/servicedocument>. Users can request this URI in web browser to get the service document for the installed Zentity AtomPub Service

#### AtomPub Post Usage

Users can upload any media/Atompub document (in AtomPub format) to any of the collection URIs (Eg. <http://MyServer:9090/AtomPub/AtomPub.ashx/Thesis>).

This sample below shows how to use AtomPub API and how to create a media resources (uploading a docx file) on the thesis collection.

* Open Microsoft Visual Studio 2008.
* Create new console application.
* Update Program.cs as shown below.

|  |
| --- |
| using System;  using System.Globalization;  using System.IO;  using System.Net;  using System.Text;  namespace ZentitySample  {  class Program  {  static void Main(string[] args)  {  HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create("http://MyServer:9090/AtomPub/AtomPub.ashx/thesis");  request.Method = "POST";  request.ReadWriteTimeout = 1000000;  request.ContentType = "application/msword";  request.AllowWriteStreamBuffering = true;  request.Credentials = CredentialCache.DefaultCredentials;  string userPassword = string.Format(CultureInfo.InvariantCulture, "{0}:{1}","UserName", "Password");  byte[] encodingData = ASCIIEncoding.ASCII.GetBytes(userPassword);  string encodedUserPassword = Convert.ToBase64String(encodingData);  request.Headers.Add(HttpRequestHeader.Authorization, "Basic " + encodedUserPassword);  int length = 64000;  byte[] byteContents = new byte[length];  FileStream fs = new FileStream(@"C:\FirstBlog.docx", FileMode.Open);  request.ContentLength = fs.Length;  int readBytes = fs.Read(byteContents, 0, length);  Stream newStream = request.GetRequestStream();  newStream.Flush();  while (readBytes != 0)  {  newStream.Write(byteContents, 0, readBytes);  readBytes = fs.Read(byteContents, 0, length);  }  fs.Close();  newStream.Close();  try  {  HttpWebResponse response =  (HttpWebResponse)request.GetResponse();  HttpStatusCode statusCode = response.StatusCode;  Stream responseStream = response.GetResponseStream();  StreamReader reader = new StreamReader(responseStream);  string responsString = reader.ReadToEnd();  reader.Close();  Console.WriteLine(responsString);  }  catch(WebException ex)  {  Console.Write(ex.Message);  }  Console.Read();  }  }  } |

Returned Atom Entry Document will be look like

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <entry xmlns="http://www.w3.org/2005/Atom">  <id>urn:guid:94010ec3-42cd-4721-989a-e5e989c0091b</id>  <title type="text" />  <summary type="text" />  <updated>2009-04-28T15:18:10+05:30</updated>  <author>  <name />  </author>  <link rel="edit-media" href="http://MyServer:9090/atompub/atompub.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b" />  <link rel="edit" href="http://MyServer:9090/atompub/atompub.ashx/Thesis/edit/94010ec3-42cd-4721-989a-e5e989c0091b" />  <content type="application/msword" src="http://MyServer:9090/atompub/ atompub.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b"/></entry> |

For posting an entry resource modify above code by changing path “C:\FirstBlog.docx” to Atom entry Document file path, and request.ContentType to “application/atom+xml;type=entry”.

#### AtomPub GET Usage

HTTP GET requests can be used for getting metadata for individual resources or a list of resources in a specific collection. The server will respond with an Atom Entry Document (in the case of GET on a resource URI) or an Atom Feed Document (in the case of GET on collection URI) for GET requests to resource URI or collection URI respectively.

1. For getting metadata for all resource in the Thesis collection:
   * Modify above code for getting service document by changing request URI to Thesis collection URI ([*http://MyServer:9090/atompub/atompub.ashx/Thesis*](http://MyServer:9090/atompub/atompub.ashx/Thesis)), and changing the request.Method to “GET”

The resulting Atom Feed document will contain an atim:entry element for each of the last 10 modified 10 resources. If this collection contains more than 10 resources, the next 10 resources can be obtained by sending a GET request to *next* link URI which can be found in the first GET request.

1. In order to retreive the metadata for a posted entry or for retrieving the binary content of a posted media resource, follow the below steps.
   * Modify above code for getting service document by changing request URI to entry resource URI (<http://MyServer:9090/atompub/atompub.ashx/Thesis/edit/94010ec3-42cd-4721-989a-e5e989c0091b>) or media resource URI (<http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b> *)* and change the request.Method to “GET”

The response Atom Entry Document will be similar to response for corresponding POST request.

#### AtomPub PUT Usage

In order to modify the posted media resource’s content and posted entry resource’s metadata, send an HTTP PUT request to the corresponding URI.

* 1. To update the resource content of a posted media resource:
* Modify the above code for posting a media resource by changing request URI to edit-media link which can be obtained from the response for the above media POST request ([http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b](http://v-anputh:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b))
* Change path “C:\FirstBlog.docx” to new media resource path.
* Change the request type to PUT.

The response Atom Entry document will be similar to the response for the above media POST request. But a GET request to the edit media link we will get updated media resource content.

* 1. In order to update the metadata of a posted media resource with a new Atom Entry Document:
* Modify the above code for posting a media resource by changing the request URI to the edit link which can be obtained from the response for the above media POST request (http://MyServer:9090/atompub/atompub.ashx/Thesis/edit/94010ec3-42cd-4721-989a-e5e989c0091b)
* Change the path “C:\FirstBlog.docx” to new Atom entry Document file path
* Change request type to PUT.

The response will contain modified Atom Entry Document.

#### AtomPub DELETE usage.

An HTTP DELETE request can be used for deleting media resource or an entry resource.

* 1. In order to delete a posted media resource:
* Modify above code for posting media resource by changing request URI to edit-media link which can be found in the response for the the above media POST request ([http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b](http://v-anputh:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b))
* Change request type to DELETE.

Server will respond with a HTTP client status code OK for successful deletion of resource.

* 1. In order to delete a posted entry resource and its media resource:
* Modify the above code for posting a media resource by changing the request URI to THE *edit link* which can be found in the response for the media POST request (http://MyServer:9090/atompub/atompub.ashx/Thesis/edit/94010ec3-42cd-4721-989a-e5e989c0091b)
* Change request type to DELETE.

Server will respond with a HTTP client status code OK for the successful deletion of resources.

### SWORD

Zentity also supports the [SWORD APP Profile 1.2](http://www.ukoln.ac.uk/repositories/digirep/index/SWORD_APP_Profile_1.2) – Level 0 for interoperable and bulk deposit between repository platforms. The SWORD service accepts only POST requests for a document of any type. When a zip file containing [METS](http://www.loc.gov/standards/mets/METSOverview.v2.html) xml data, is posted to the SWORD service, the zip file’s contents are extracted and individual files are uploaded to the repository.

#### SWORD POST usage

This sample shows how to use the SWORD APIs and SWORD service to create, edit or retrieve resources.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add references to System.Data.Entity and Zentity.Core, Zentity.Platform (which can be found at <installation path>\Platform folder)
* Add an application configuration file with connection string as mentioned [here](#_Creating_basic_console)
* Update Program.cs as shown below

|  |
| --- |
| using System;  using System.Net;  using System.IO;  using System.Text;  using System.Globalization;  namespace ZentitySample  {  class Program  {  static void Main(string[] args)  {    HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create("http://MyServer:9090/AtomPub/Sword/Sword.ashx/thesis");  request.Method = "POST";  request.ReadWriteTimeout = 1000000;  request.ContentType = "application/zip";  request.AllowWriteStreamBuffering = true;  request.Credentials = CredentialCache.DefaultCredentials;  string userPassword = string.Format(CultureInfo.InvariantCulture, "{0}:{1}", "UserName", "Password");  byte[] encodingData = ASCIIEncoding.ASCII.GetBytes(userPassword);  string encodedUserPassword = Convert.ToBase64String(encodingData);  request.Headers.Add(HttpRequestHeader.Authorization, "Basic " + encodedUserPassword);  int length = 64000;  byte[] byteContents = new byte[length];  FileStream fs = new FileStream(@"E:\FirstBlog.zip", FileMode.Open);  request.ContentLength = fs.Length;  int readBytes = fs.Read(byteContents, 0, length);  Stream newStream = request.GetRequestStream();  newStream.Flush();  while (readBytes != 0)  {  newStream.Write(byteContents, 0, readBytes);  readBytes = fs.Read(byteContents, 0, length);  }  fs.Close();  newStream.Close();  try  {  HttpWebResponse response =  (HttpWebResponse)request.GetResponse();  HttpStatusCode statusCode = response.StatusCode;  Stream responseStream = response.GetResponseStream();  StreamReader reader = new StreamReader(responseStream);  string responsString = reader.ReadToEnd();  reader.Close();  }  catch(WebException ex)  {  Console.Write(ex.Message);  }  Console.Read();  }  }  } |

The Atom Entry Document returned is formatted like,

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <entry xmlns="http://www.w3.org/2005/Atom">  <id>urn:guid:94010ec3-42cd-4721-989a-e5e989c0091b</id>  <title type="text" />  <summary type="text" />  <updated>2009-04-28T15:18:10+05:30</updated>  <author>  <name />  </author>  <link rel="edit-media" href="http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b" />  <link rel="edit" href="http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit/94010ec3-42cd-4721-989a-e5e989c0091b" />  <content type="application/zip" src="http://MyServer:9090/atompub/sword/sword.ashx/Thesis/edit-media/94010ec3-42cd-4721-989a-e5e989c0091b" />  <treatment xmlns="http://purl.org/net/sword/">Successfully created a thesis</treatment>  </entry> |

## Platform API

Zentity Platform provides developer friendly APIs for building custom applications.

### Search

Zentity provides a search interface to retrieve information from the repository resources, and which leverages the built in full-text indexing of SQL Server 2008. Zentity Search also provides an end-user query syntax which is similar to Advanced Query Syntax (AQS) and allows users to perform more focused property based searches beyond simple keyword search.

Zentity Search has three main components: Parser, Interpreter & Search Engine.

**Parser** takes a logical search query, tokenizes it, parses it and validates it to create a parse tree. The parse tree is the internal representation of the search criteria that is passed through the search query string.

**Interpreter** is responsible for converting the output of parser (a parse tree) into a TSQL query. While doing so the module makes decisions on using a Full Text Search or a TSQL query for string searches.

**Search Engine** is a thin layer exposed as public API. This accepts a search query, gets parse tree by using Parser, hands over parse tree to Interpreter which returns TSQL and then communicates with the database to retrieve the entity objects by supplying the TSQL.

#### Example usage of Zentity Search

This sample below shows how to use Zentity Search to by supplying a logical search string.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add following .NET references
  + System.Data.Entity,
  + System.Configuration
* Add following Zentity references which can be found at <installation path>\Platform folder
  + Zentity.Platform
  + Zentity.Core (core library for managing resources and their associations)
  + Zentity.ScholarlyWorks (to use built-in resource types from Scholarly works data model and relationships between them)
* Add an application configuration file with connection string as mentioned [here](#_Quick_usage_of)
* Update Program.cs as shown below

|  |
| --- |
| using System;  using System.Data.SqlClient;  using System.Configuration;  using System.Collections.Generic;  using Zentity.Core;  using Zentity.Platform;  using Zentity.ScholarlyWorks;  namespace ZentitySamples  {  class Program  {  static void Main(string[] args)  {  try  {  using(ZentityContext context = new ZentityContext())  {  // Load ScholarlyWork metadata.  context.MetadataWorkspace.LoadFromAssembly(typeof(ScholarlyWork).Assembly);    string searchQuery = "Title:\"Quantum Universe\"";  // Specify the page size.  int pageSize = 10;  SearchEngine search = new SearchEngine(pageSize);    int currentCursor = 0;  int totalMatchingRecords;    // Retrieve 1st 10 matching records.  IEnumerable<Resource> matchingItems =  search.SearchResources(  searchQuery, context, currentCursor, out totalMatchingRecords);    Console.WriteLine("Total matching items: {0}", totalMatchingRecords);    foreach (Resource item in matchingItems)  {  Console.WriteLine(  string.Format("\nId = {0} \nTitle = {1}", item.Id, item.Title));  }    // If there are more number of records that are to be fetched  // then call Search function again to retrieve the next set  // of results.  currentCursor += pageSize;  while (currentCursor < totalMatchingRecords)  {  matchingItems =  search.SearchResources(  searchQuery, context, currentCursor, out totalMatchingRecords);  foreach (Resource item in matchingItems)  {  Console.WriteLine(  string.Format("\nId = {0} \nTitle = {1}", item.Id, item.Title));  }  currentCursor += pageSize;  }  }  }  catch (ArgumentNullException exception)  {  Console.WriteLine("Exception : {0}", exception.Message);  }  catch (ArgumentException exception)  {  Console.WriteLine("Exception : {0}", exception.Message);  }  catch (SearchException exception)  {  Console.WriteLine("Exception : {0}", exception.Message);  }  catch (ConfigurationErrorsException exception)  {  Console.WriteLine("Exception : {0}", exception.Message);  }  catch (SqlException exception)  {  Console.WriteLine("Exception : {0}", exception.Message);  }  }  }  } |

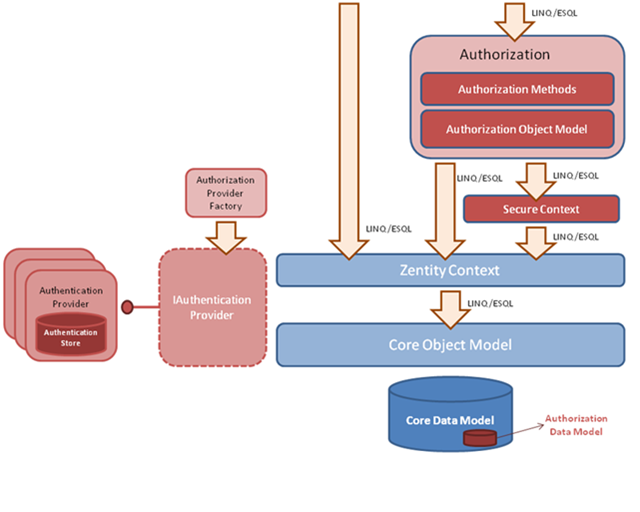
### Security

Zentity provides a pluggable security model to manage Users/Groups using authentication and restrict permissions (Read/Update/Delete/Owner) on resources for users & groups using Authorization.

**Authentication** – This Pluggable module contains a contract which can be implemented to support different authentication types like Windows Live ID, integrated Windows Authentication, Active Directory, certificate or custom authentication implementation. It provides an implementation of the contract along with user management features. Also, this module is independent of the Zentity storage model.

**Authorization** – This Zentity-Core based module contains *a data model created using Core extensibility API,* a secure context to authorize DML operations, and LINQ extension methods to integrate authorization calls with LINQ queries to Core. It stores authorization data in the form of relationships using special predicates.

**Authorization Helper** – This layer provides security logic on top of the authentication and authorization layers, and shows one of the ways security base APIs can be used.



*Diagram – Authentication and Authorization API’s.*

Authorization Data Model

Core Data Model

Core Object Model

Authentication API

Authorization API

TSQL Authorization

User Management

Permission Management

Authorization

Authorization Helper

*Diagram – Authorization Helper API.*

The security design is based on an ‘opt-in’ model, which allows a component developer to make his component secure, but does not mandate using security. The model also provide data access layer which provides various methods to mange user.

#### Security API usage

This sample below shows how to use create users and groups.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add references to
  + System.Data.Entity
  + System.IdentityModel
  + Zentity.Core
  + Zentity.Security.Authentication
  + Zentity.Security.AuthenticationProvider
  + Zentity.Security.Authorization
  + Zentity.Security.AuthorizationHelper
* Add an application configuration file similar to the one given [here](#SecurityConfig).
* Update Program.cs as shown below
* Replace administrator password with the correct one for your installation.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using Zentity.Security.AuthorizationHelper;  using Zentity.Core;  using Zentity.Security.AuthenticationProvider;  using Zentity.Security.Authentication;  using System.IdentityModel.Tokens;  using Zentity.Security.Authorization;  namespace SecuritySamples  {  class Program  {  static void Main(string[] args)  {  //An administrator can create new users. Get an instance of authentication provider and login as administrator.  //The 'ZentityAuthenticationProvider' is configured in the application configuration file. Please refer to the sample  //application configuration file provided in the guide.  IAuthenticationProvider provider = AuthenticationProviderFactory.CreateAuthenticationProvider("ZentityAuthenticationProvider");  AuthenticatedToken adminToken = provider.Authenticate(new UserNameSecurityToken("Administrator", "XXXX")); //Supply correct password    //Create a ZentityUser instance and set its properties.  ZentityUser user = new ZentityUser  {  LogOnName = "John", //mandatory  FirstName = "John", //mandatory  LastName = "King",  City = "New York",  Country = "USA",  Email = "john@abc.com" //mandatory  };  user.SetPassword("john@123");//mandatory  user.SetSecurityQuestion("??");//mandatory  user.SetAnswer("\*\*");//mandatory  //You need an instance of ZentityContext for creating the user in authorization (core) database.  using (ZentityContext context = new ZentityContext())  {  if (UserManager.CreateUser(user, adminToken))  {  Console.WriteLine("User John created");  }  else  {  Console.WriteLine("Errors while creating user.");  }  //Create a group  Group users = new Group { GroupName = "Users", Title = "Users", Description = "This is a users group." };  if (UserManager.CreateGroup(users, adminToken))  {  Console.WriteLine("Group created.");  }  else  {  Console.WriteLine("Errors while creating group.");  }  //Add John to users group.  Identity john = UserManager.GetIdentity("John", context);  if (UserManager.AddIdentityToGroup(john, users, adminToken))  {  Console.WriteLine("John added to users group.");  }  else  {  Console.WriteLine("Errors while adding John to users group.");  }  }  }  }  } |

This sample below shows how to grant permissions and authorize operations.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add references to
  + System.Data.Entity
  + System.IdentityModel
  + Zentity.Core
  + Zentity.Security.Authentication
  + Zentity.Security.AuthenticationProvider
  + Zentity.Security.Authorization
  + Zentity.Security.AuthorizationHelper
* Add an application configuration file similar to the one given [here](#SecurityConfig).
* Update Program.cs as shown below
* Replace administrator password with the correct one for your installation. It appears at two places in this sample.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Zentity.Security.AuthorizationHelper;

using Zentity.Core;

using Zentity.Security.AuthenticationProvider;

using Zentity.Security.Authentication;

using System.IdentityModel.Tokens;

using Zentity.Security.Authorization;

namespace SecuritySamples

{

class Program

{

static void Main(string[] args)

{

CreateUsersAndGroups();

//Login as Admin

IAuthenticationProvider provider = AuthenticationProviderFactory.CreateAuthenticationProvider("ZentityAuthenticationProvider");

AuthenticatedToken adminToken = provider.Authenticate(new UserNameSecurityToken("Administrator", "XXXX")); //Supply correct password

//Grant create access to Mary

using (ZentityContext context = new ZentityContext())

{

context.MetadataWorkspace.LoadFromAssembly(typeof(Identity).Assembly);

Identity Mary = UserManager.GetIdentity("Mary", context);

if (Mary.GrantCreate(adminToken, context))

{

Console.WriteLine("Create access granted to Mary.");

}

//Security APIs do not save context changes. Hence the following call is necessary after calling grant/revoke/remove.

context.SaveChanges();

}

//Create resources and grant permissions on them.

//Login as Mary

AuthenticatedToken marysToken = provider.Authenticate(new UserNameSecurityToken("Mary", "Mary@123"));

using (ZentityContext context = new ZentityContext())

{

context.MetadataWorkspace.LoadFromAssembly(typeof(Identity).Assembly);

//Before allowing Mary to create new resources, the application should check whether Mary has create access.

if (marysToken.HasCreatePermission(context))

{

Resource r = new Resource { Title = "Resource 1", Description = "Mary created this resource" };

context.AddToResources(r);

context.SaveChanges();

//Make Mary owner of the resource. This is an explicit step which the application needs to take when

//it allows a user to create a resource.

r.GrantDefaultPermissions(context, marysToken);

context.SaveChanges();

//Now Mary wants to grant update on this resource to his collegue Patrick.

//Since he is owner of the resource he can grant / revoke / remove permissions of other users on his resource.

Identity patrick = UserManager.GetIdentity("Patrick", context);

r.Grant("Update", patrick, context, marysToken);

//Security APIs do not save context changes. Hence the following call is necessary after calling grant/revoke/remove.

context.SaveChanges();

}

}

//Now Patrick wants to update the resource on which Mary has granted him update.

//Login as Patrick.

AuthenticatedToken patricksToken = provider.Authenticate(new UserNameSecurityToken("Patrick", "patrick@123"));

using (ZentityContext context = new ZentityContext())

{

context.MetadataWorkspace.LoadFromAssembly(typeof(Identity).Assembly);

Resource r = context.Resources.Where(res => res.Title.Equals("Resource 1")).FirstOrDefault();

if (r != null)

{

//Check whether the logged on user has update access.

if (r.Authorize("Update", context, patricksToken))

{

Console.WriteLine("Patrick has update access on Resource 1.");

}

else

{

Console.WriteLine("Patrick does not have update access on Resource 1.");

}

}

}

}

private static void CreateUsersAndGroups()

{

//An administrator can create new users. Get an instance of authentication provider and login as administrator.

//The 'ZentityAuthenticationProvider' is configured in the application configuration file. Please refer to the sample

//application configuration file provided in the guide.

IAuthenticationProvider provider = AuthenticationProviderFactory.CreateAuthenticationProvider("ZentityAuthenticationProvider");

AuthenticatedToken adminToken = provider.Authenticate(new UserNameSecurityToken("Administrator", "XXXX")); //Supply correct password

//Create a ZentityUser instance and set its properties.

ZentityUser Mary = new ZentityUser

{

LogOnName = "Mary", //mandatory

FirstName = "Mary", //mandatory

LastName = "King",

City = "New York",

Country = "USA",

Email = "Mary@abc.com" //mandatory

};

Mary.SetPassword("Mary@123");//mandatory

Mary.SetSecurityQuestion("??");//mandatory

Mary.SetAnswer("\*\*");//mandatory

//Create a ZentityUser instance and set its properties.

ZentityUser patrick = new ZentityUser

{

LogOnName = "Patrick", //mandatory

FirstName = "Patrick", //mandatory

LastName = "Smith",

City = "New York",

Country = "USA",

Email = "patrick@abc.com" //mandatory

};

patrick.SetPassword("patrick@123");//mandatory

patrick.SetSecurityQuestion("??");//mandatory

patrick.SetAnswer("\*\*");//mandatory

//You need an instance of ZentityContext for creating the user in authorization (core) database.

using (ZentityContext context = new ZentityContext())

{

context.MetadataWorkspace.LoadFromAssembly(typeof(Identity).Assembly);

if (UserManager.CreateUser(Mary, adminToken))

{

Console.WriteLine("User Mary created");

}

else

{

Console.WriteLine("Errors while creating user.");

}

//Create a group

Group users = new Group { GroupName = "Group1", Title = "Group1", Description = "This is a users group." };

if (UserManager.CreateGroup(users, adminToken))

{

Console.WriteLine("Group created.");

}

else

{

Console.WriteLine("Errors while creating group.");

}

//Add Mary to users group.

Identity user = UserManager.GetIdentity("Mary", context);

if (UserManager.AddIdentityToGroup(user, users, adminToken))

{

Console.WriteLine("Mary added to Group1.");

}

else

{

Console.WriteLine("Errors while adding Mary to Group1.");

}

//Create user Patrick

if (UserManager.CreateUser(patrick, adminToken))

{

Console.WriteLine("User Patrick created");

}

else

{

Console.WriteLine("Errors while creating user.");

}

}

}

}

#### }

#### Sample application configuration file for Security

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<!-- This section adds reference to the assemblies containing security configuration sections. -->

<section name="Authentication" type="Zentity.Security.Authentication.AuthenticationConfigurationSection,

Zentity.Security.Authentication"/>

<section name="PasswordPolicy" type="Zentity.Security.AuthenticationProvider.PasswordManagement.PasswordPolicy,

Zentity.Security.AuthenticationProvider"/>

</configSections>

<!--

Authentication configuration section

Add one provider entry for each authentication implementation.

name = Choose a unique name for authentication provider.

type = Name of the type implementing IAuthenticationProvider, Fully qualified assembly name

-->

<Authentication>

<Providers>

<add name="ZentityAuthenticationProvider" type="Zentity.Security.AuthenticationProvider.ZentityAuthenticationProvider,

Zentity.Security.AuthenticationProvider"/>

</Providers>

</Authentication>

<!--

Password policy configuration section. Please add only one policy entry in this section. Otherwise the behavior of the application cannot

be predicted.

MinimumLength = Desired minimum length of the password. Value must be >= 6

MaximumLength = Desired maximum length of the password. Value must be <= 100.

ExpiresInDays = Number of days after which a user must change his password. Value must be between 7 and 100.

StartWithAlphabet = Set to True if password must start with an alphabet.

MustContainDigit = Set to True if password must contain a digit.

MustContainSpecialCharacter = Set to True if password must contain a special character - ~`!@#$%^&\*()\_-+={[}]|\\:;\"'<,>.?/

-->

<PasswordPolicy>

<CurrentPolicy

MinimumLength="7"

MaximumLength="20"

ExpiresInDays="90"

StartWithAlphabet="true"

MustContainDigit="true"

MustContainSpecialCharacter="true"/>

</PasswordPolicy>

<connectionStrings>

<!--Connection string for the authentication and authorization databases-->

<!-- This connection string is required for accessing core and authorization data models. -->

<add name="ZentityContext" connectionString="metadata=res://Zentity.Security.Authorization;

provider=System.Data.SqlClient;

provider connection string=&quot;Data Source=.;

Initial Catalog=Zentity;

Integrated Security=True;

MultipleActiveResultSets=True&quot;"

providerName="System.Data.EntityClient"/>

<!-- This connection string is required for accessing core, authorization and scholarlyworks data models. -->

<add name="ScholarlyWorksConnection" connectionString="metadata=res://Zentity.ScholarlyWorksAndAuthorization;

provider=System.Data.SqlClient;

provider connection string=&quot;Data Source=.;

Initial Catalog= Zentity;

Integrated Security=True;

MultipleActiveResultSets=True&quot;"

providerName="System.Data.EntityClient"/>

<!-- This connection string is required for accessing the authentication database. -->

<add name="AuthenticationConnection" connectionString="Data Source=.;Initial Catalog=ZentityAuthentication; integrated security=true"/>

<!-- This connection string is required for opening a SqlConnection with the core database. (This database should also have authorization

data model installed. Zentity installer installs the core database with scholarlyworks and authorization data models.) -->

<add name="CoreSqlConnection" connectionString="Data Source=.;

Initial Catalog=Zentity;

Integrated Security=True;

MultipleActiveResultSets=True"/>

</connectionStrings>

<appSettings>

<!--

This configuration setting decides whether the password policy is applied. If this is set to true

the new passwords entered while changing password or creation of user account will be checked against the password policy.

-->

<add key="ApplyPasswordPolicy" value="False"/>

<!--

IMPORTANT It is recommended that this setting should be set to true from the point of view of ensuring secure storage of passwords.

This configuration setting can be changed only once right after Zentity is installed, before creating any

user accounts. If this is changed afterwards it will lead to errors in authenticating users.

Also the built in users - administrator and guest have their passwords stored in encrypted form. You need to set them in clear text in

the authentication database, if this setting is made false.

-->

<add key="EncryptPassword" value="True"/>

<add key="encryptionKey" value="877D3C2437D7B9DF4F31E9AE6CA97C3F8AE99B02BF66D137F865A8651C1A36D2"/>

</appSettings>

</configuration>

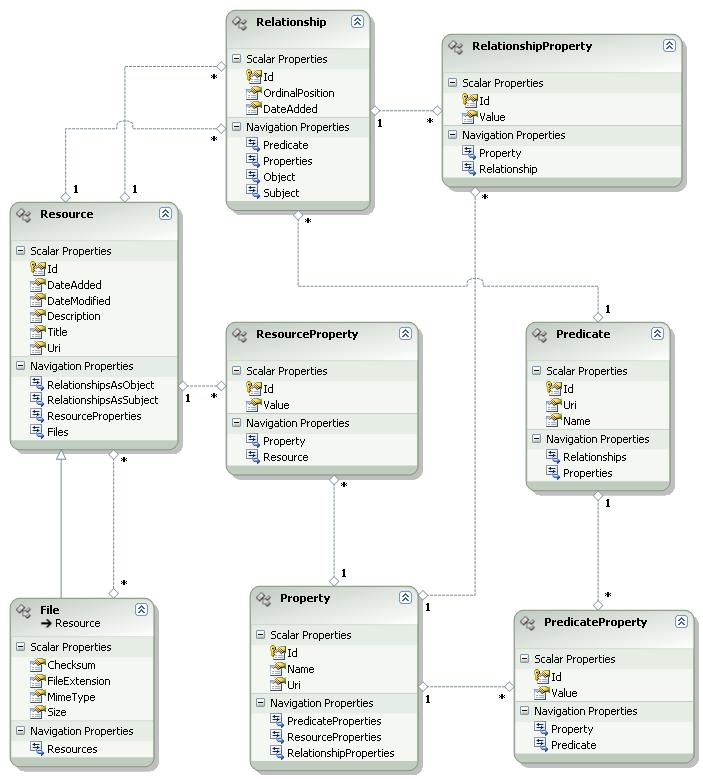
## Data Models

Zentity provides a Core data model & the domain specific built-in Scholarly Works data model.

### Core data model

The Core Entity Data Model provides a foundation with high-level entities such as Resource, Predicate, Property, File and Relationship. Developers can use these entities as well as extend them to build a custom data model based on semantic relationships.

The Core API has basic entities defined using Entity Framework and uses SQL Server 2008 as information store with [Filestream](http://msdn.microsoft.com/en-us/library/cc949109.aspx) for Content File storage.



#### Creating basic console application

This sample below shows how to create resources and relationships using core entities.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add references to System.Data.Entity, System.Runtime.Serialization and Zentity.Core (core library for managing resources and their associations, which can be found at <installation path>\Platform folder)
* Add an application configuration file with connection string as shown below *(This configuration file entry is required for all the samples provided in this document)*

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <configuration>  <connectionStrings>  <add name="ZentityContext"  connectionString="provider=System.Data.SqlClient;  metadata=res://Zentity.Core;  provider connection string='Data Source=.;  Initial Catalog=Zentity;Integrated Security=True;'"  providerName="System.Data.EntityClient" />  </connectionStrings>  </configuration> |

* Update Program.cs as shown below

|  |
| --- |
| using Zentity.Core;  using System;  using System.Linq;  namespace ZentitySamples  {  class Program  {  static void Main(string[] args)  {  Guid resourceId;  using (ZentityContext context = new ZentityContext())  {  Resource lecture = new Resource { Title = "The Lecture" };  Resource speaker = new Resource { Title = "The Author" };  File presentation = new File { Title = "The Presentation" };  // Add file to lecture.  lecture.Files.Add(presentation);  // Add a custom relationship.  Predicate lectureDeliveredBy = new Predicate  {  Name = "DeliveredBy",  Uri = "urn:" + Guid.NewGuid().ToString("N")  };  Relationship triple = new Relationship  {  Subject = lecture,  Object = speaker,  Predicate = lectureDeliveredBy  };  context.AddToResources(lecture);  context.SaveChanges();  resourceId = lecture.Id;  }  using (ZentityContext context = new ZentityContext())  {  foreach (Relationship relationship in  context.Relationships.Include("Subject")  .Include("Object").Include("Predicate"))  {  if (relationship.Subject != null  && relationship.Subject.Id == resourceId)  {  Console.WriteLine(  string.Format("Triple: {0}>>-{1}->>{2}",  relationship.Subject.Title,  relationship.Predicate.Name,  relationship.Object.Title));  }  }  }  }  }  } |

### Scholarly works data model

Zentity also provides a Scholarly Works data model which is derived from the Core Data Model. This also showcases how one can create domain specific custom data model.

The ScholarlyWorksItem is derived from the Core Resource entity and all subsequent entities are derived from the same. This model provides built-in entities such as Lecture, Publication, Paper, Presentation, Communication, Media, File, Contact, and Tag along with basic properties for each of these and well known relationships such as Author, Cites, Version and so on.

*Details about each entity, its Properties & Associations can be found in help file (Zentity.Core.chm).*

#### Quick usage of Scholarly works entities

This sample below shows how to use the built-in resource types from Scholarly works data model and the relationships between them.

* Open Microsoft Visual Studio 2008,
* Create new console application,
* Add references to System.Data.Entity, System.Runtime.Serialization
* Add references to Zentity.Core & Zentity.ScholarlyWorks (which can be found at <installation path>\Platform folder)
* Add an application configuration file with connection string as shown below

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <configuration>  <connectionStrings>  <add name="ZentityContext"  connectionString="provider=System.Data.SqlClient;  metadata=res://Zentity.ScholarlyWorks;  provider connection string='Data Source=.;  Initial Catalog=Zentity;Integrated Security=True;'"  providerName="System.Data.EntityClient" />  </connectionStrings>  </configuration> |

* Update Program.cs as shown below

|  |
| --- |
| using Zentity.Core;  using System;  using System.Linq;  using Zentity.ScholarlyWorks;  namespace ZentitySamples  {  class Program  {  static void Main(string[] args)  {  Guid resourceId;  using (ZentityContext context = new ZentityContext())  {  Publication aPublication = new Publication { Title = "A Publication" };  Person anAuthor = new Person { FirstName = "Bob" };  // Associate publication and author.  aPublication.Authors.Add(anAuthor);  context.AddToResources(aPublication);  context.SaveChanges();  resourceId = aPublication.Id;  }  using (ZentityContext context = new ZentityContext())  {  Publication aPublication = context.Publications().  Where(tuple => tuple.Id == resourceId).First();  aPublication.Authors.Load();  foreach (Person p in aPublication.Authors)  {  Console.WriteLine("{0} <--is authored by--> {1}", aPublication.Title, p.FirstName);  }  }  }  }  } |

# More Features

Other Zentity features in Version 2 release are demonstrated in the Zentity Version 2 – Getting Started Guide

Details on other features are mentioned in the CHM help files installed as part of the product.