****

Styx – the Search Service

Search service

Findwise Stockholm

2013-2-06

Table of Contents

Introduction 6

Architecural overview 7

Diagram 7

Findwise.Styx.Core.Wcf.Service 8

Findwise.Styx.Core 8

Findwise.Styx.Api 10

Consuming the service 10

Endpoints 10

Search service 10

Admin service 10

Log analyzer service 10

Operations 10

Deployment 11

Running as an console application 11

Running as an windows service 11

Configuration 12

App.config 12

Logging 12

Admin GUI 13

Consumer overview 13

Edit/Add/Delete Consumer 13

Information context overview 14

Edit/Add/Delete Information context 14

Facets overview 15

Edit/Add/Delete Facet 15

Technologies 16

Windows Communication Foundation (WCF) 16

Windsor Castle 16

Topshelf 16

Log4Net 16

MongoDB 16

RavenDB 16

Dapper 17

The RestClient 18

Example code 19

Using the client 19

Creating searchers 19

# Introduction

The purpose of the search service is to respond to search requests with search results. It also provides handy information about how to formulate a search request. The service is built using Microsoft WCF with .Net framework 4.0.

The main function of the service is to listen to the uri template “Search/\*” for POST requests formulated either in xml or json.

This document will describe the journey of a request up until the response is finally returned to the consumer.

# Architecural overview

## DiagramC:\1.Jobb\1.Managed\styx\Documentation\Styx architecture.jpg

## Findwise.Styx.Core.Wcf.Service

This project holds the web controller that listens for requests. The service is defined by the ISearchService interface for which there is one implementation, SearchService.

The SearchService will upon receiving a POST request with xml or json try to de-serialize the request into a SearchRequest object. If the serialization fails an exception will be thrown as well as the proper HTTP status code returned to the consumer.

Assuming a successful de-serialization into a SearchRequest, the first task of the SearchService is to look up the consumer based on the ApiKey element of the SearchRequest. If none is found, an error will be thrown explaining that there was no Consumer found for the ApiKey specified. Consumers are persisted in a persistence store (mongoDB or RavenDb) and have a default instance with the ApiKey “default” for testing purposes. The default Consumer will only have access to test data.

The search service then delegates to an implementation of ISearcherManager that accepts a SearchRequest as well a SearchContext which holds the user issuing the request as well as the Consumer.

## Findwise.Styx.Core

This is the main project of Styx where most of the business logic of the application as well as necessary interfaces can be found.

The entry-point of the project is the ISearcherManager. Responsibilities of the ISearcherManager implementation:

* Logging of requests and responses
* Delegating to the correct implementation of ISearcherExecutor
* Apply implementations of ISearchProcessorService
* Associating the request with the correct InformationContext for that consumer

It is noteworthy to mention that an implementation of IInformationContextManagerService is used to merge possibly multiple InformationContext’s into one. The default implementation is to merge so that all restrictions from all InformationContexts is applied.

Next in line in the journey of a SearchRequest is the actual execution. This is handled by ISearcherExectutor which defines two functions. Firstly there is a CanHandle function which simply answers true or false given a Class. This is what is used when deciding which of many SearcherExecutor implementations to use. Secondly there is ExecuteSearch which accepts two parameters, the Searcher which is to be executed as well as an instance of InformationContext which represents the context in which the Searcher should be executed in.

Depending on what search engine styx should consume different implementation of the ISearcherExecutor should be used. A typical implementation uses an IQueryForXmlRequest implementation which given a complete URL queries it and builds an XDocument from the response which is returned. With that XDocument a implementation of IResponseInterpreter built for the specific search engine can be used to turn the XDocument into a proper SearcherResult. It is the responsibility of the ISearcherExecutor implementation to set both EngineExecutionTime as well as Searcher implementation specific properties such as QueryUsed.

Once the SearcherExecutor implementation is finished it sends the result upwards in the call chain to finally land back with the Wcf service which serializes the response and writes it to the original POST request.

## Findwise.Styx.Api

# Consuming the service

By default, **Styx** exposes two endpoints over HTTP: one for admin related tasks (modifying consumers and information contexts) and one for the actual search functions.

## Endpoints

By default Styx supports web messaging with either the xml or json dialect.

### Search service

* Default base address: [**http://localhost:1337/***[xml/json]***/**](http://localhost:1337/%5bxml/json%5d/)

Exposes all functions needed to make a query. All operations require that a valid **apiKey** is included (either in the SearchRequest or as a query parameter).

### Admin service

* Default base address**: http://localhost:1338/***[xml/json]***/**

Used for administrating available consumers; their information contexts and which facets are available.

### Log analyzer service

* Default base address**: http://localhost:1339/***[xml/json]***/**

Exposing functions that relates to analyzing the query and click logs.

## Operations

Each endpoint features a help page where all available operation (functions) are listed included what HTTP verb (GET/POST) is needed to invoke it.

# Deployment

Styx can either be run as a console application or as a windows service. Note that the executing user must have privileges to expose endpoints (open a port), else the application will fail to start.

The actual deploy procedure is a simple copy + paste of all the files.

## Running as an console application

Double click **Findwise.Styx.Topshelfhost.exe** (you might need to right click the file and *run as administrator*). All output can be seen in the console window.

## Running as an windows service

Execute **Findwise.Styx.Topshelfhost.exe install** from command line. If the installation was successful you will see a new service (default *Findwise.Styx*, can be changed in the configuration file) In window’s service manager.

# Configuration

Configuration can be done in two files: log4net.config and the app.config.

## App.config

* Findwise.Styx.TopShelfHost.config

The entries in this file are used for all application logic, such as where the search engine’s entry point, information about hit serialization, how to persist data (mongoDB, ravenDB, SQL Server etc.), endpoint exposure etc.

## Logging

* *\log4net.config*

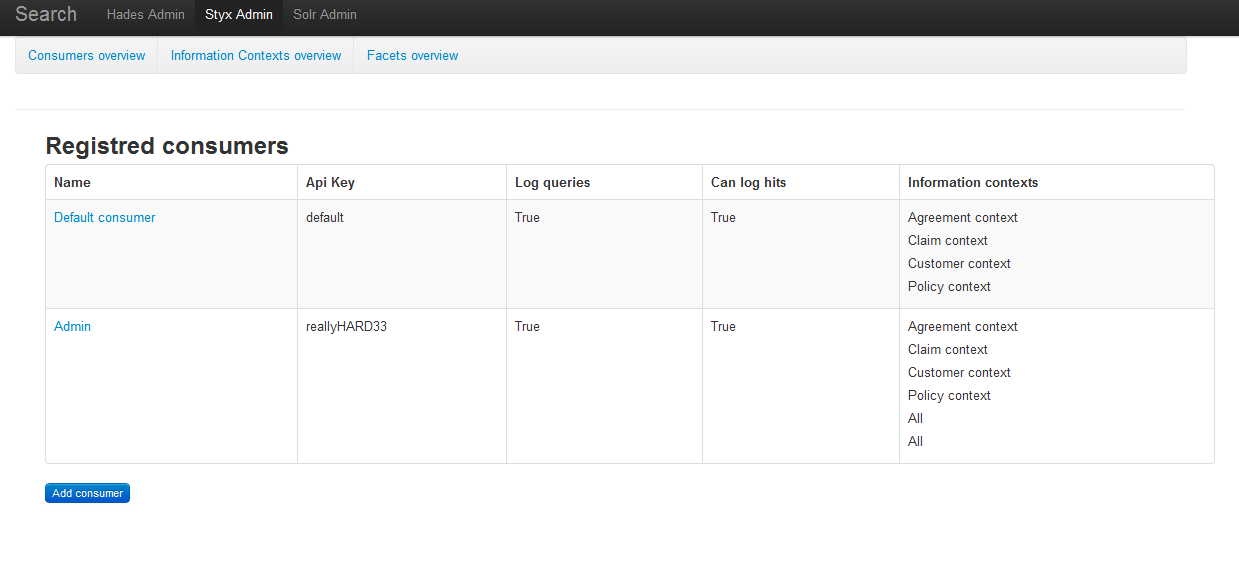
Handles logging settings, such as log appenders, log levels and presentation format.

# Admin GUI

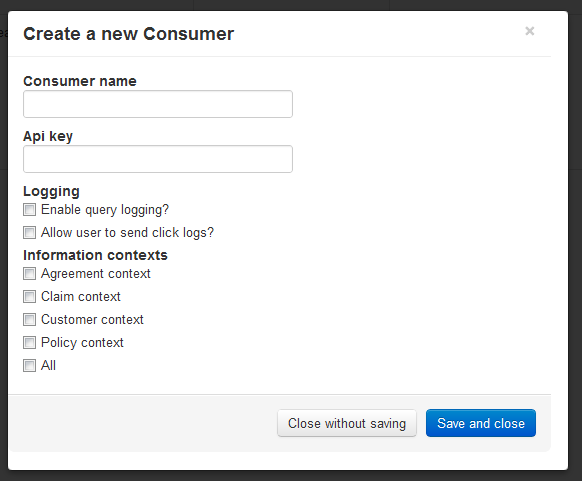
Along with Styx, an mvc.net 3 application is shipped. When deployed and configured to consume a Styx service, Consumers, Information Contexts and Facets can be created/edited/deleted. Making a change in the admin UI will send service calls to Styx that persist any changes made.

## Consumer overview

In order to add or edit an existing consumer, click the consumer name.

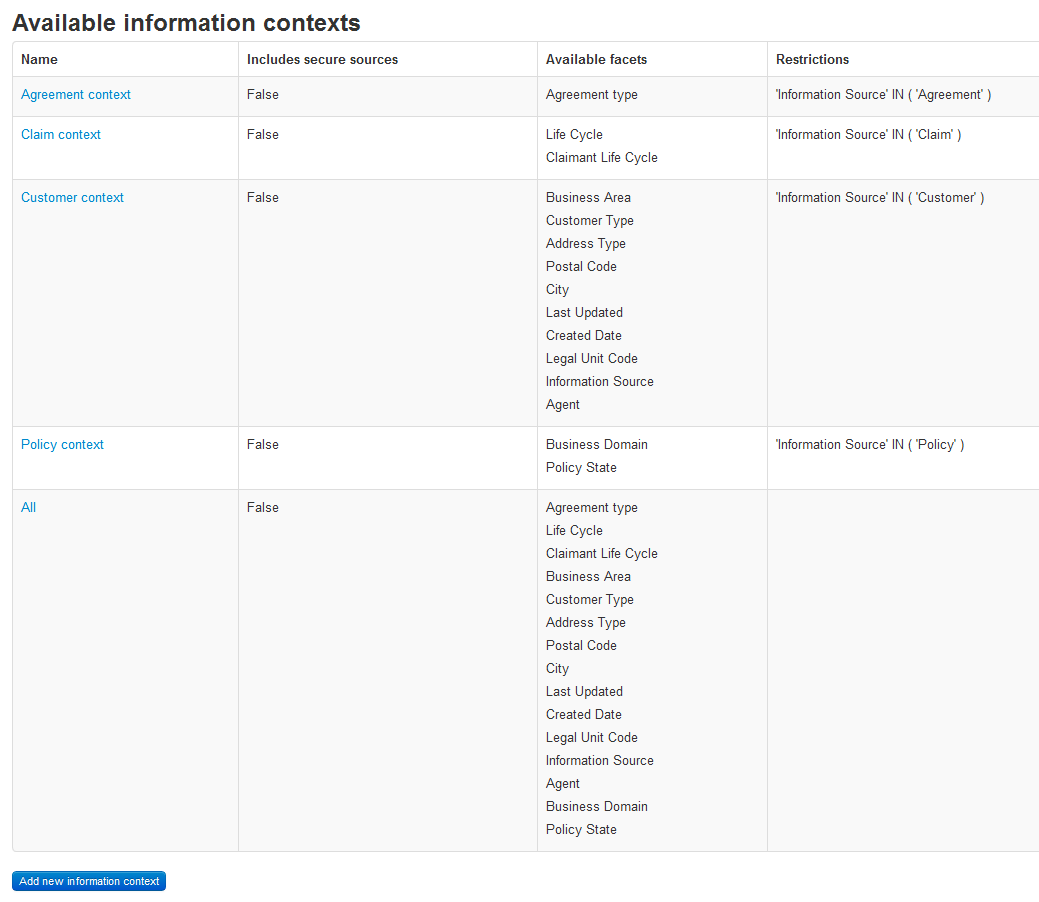


### Edit/Add/Delete Consumer



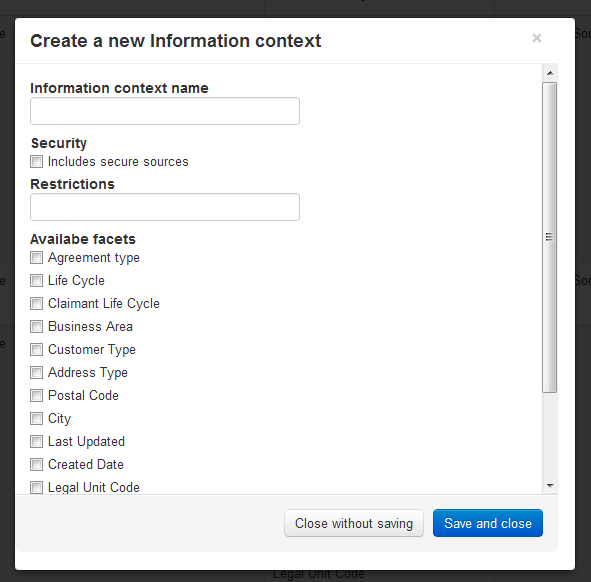
## Information context overview

In order to edit or delete an existing context, just click the context’s name.



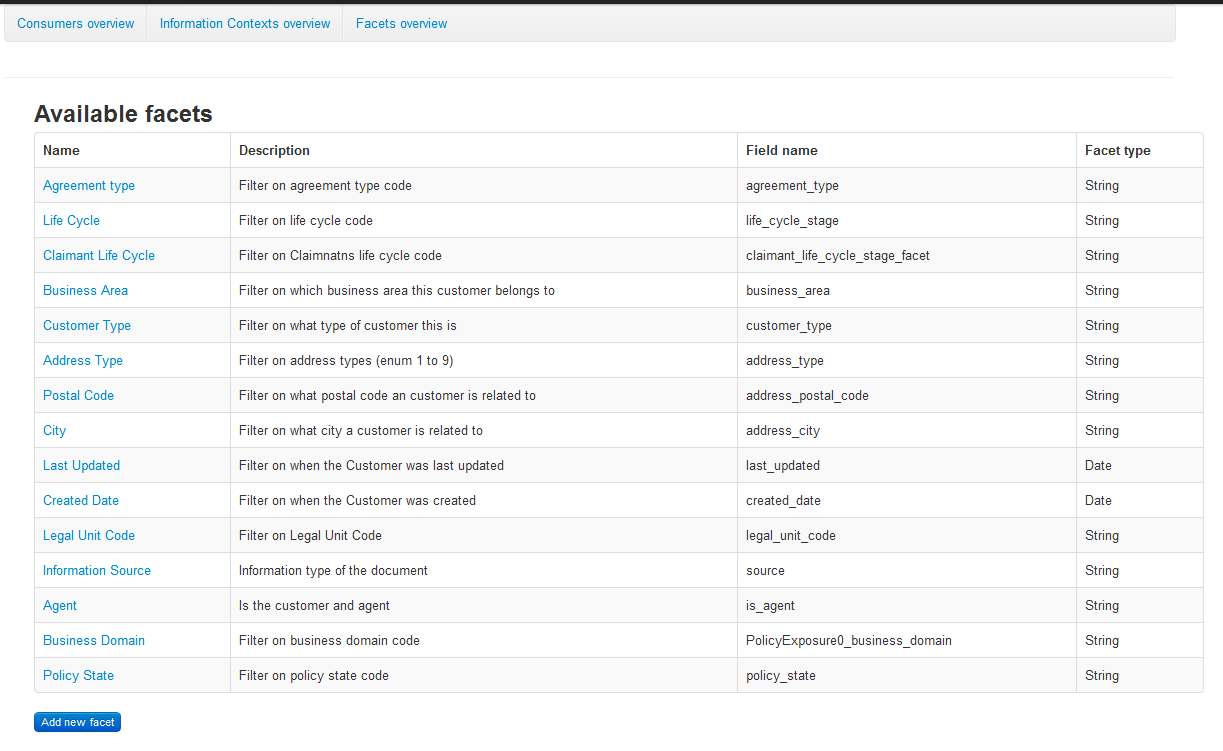
### Edit/Add/Delete Information context

So far “include secure sources” does nothing since this functionality isn’t implemented in Styx.

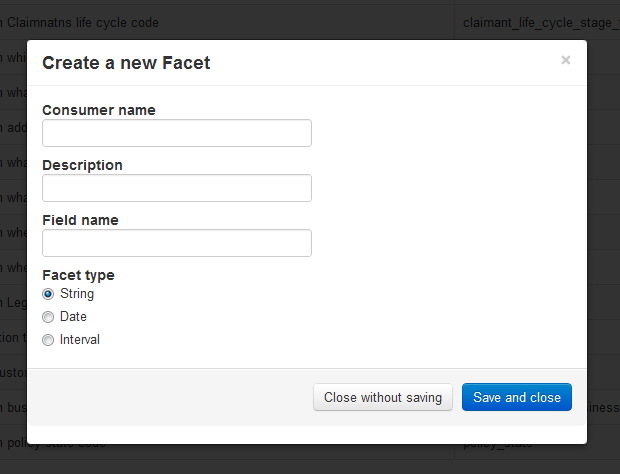


## Facets overview

Click a facet’s name to edit or delete it.



### Edit/Add/Delete Facet



# Technologies

## Windows Communication Foundation (WCF)

* <http://msdn.microsoft.com/en-us/netframework/aa663324>

a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications that communicate across the web and the enterprise.

## Windsor Castle

* <http://stw.castleproject.org/Windsor.MainPage.ashx>

The most mature inversion of control container around.

## Topshelf

* <http://topshelf-project.com/>

An easy service hosting framework for building Windows services using .NET or Mono.

## Log4Net

* <http://logging.apache.org/log4net/>

The Apache log4net library is a tool to help the programmer output log statements to a variety of output targets. log4net is a port of the Apache log4j framework.

## MongoDB

* [http://mongodb.org](http://logging.apache.org/log4net/)

MongoDB is part of the [NoSQL](http://en.wikipedia.org/wiki/NoSQL) family of database systems. Instead of storing data in tables as is done in a "classical" relational database, MongoDB stores structured data as [JSON](http://en.wikipedia.org/wiki/JSON)-like documents with dynamic schemas (MongoDB calls the format [BSON](http://en.wikipedia.org/wiki/BSON)), making the integration of data in certain types of applications easier and faster.

## RavenDB

* <http://ravendb.net/>

RavenDB is a **transactional**, **open-source** [Document Database](http://ravendb.net/docs/2.0/intro/what-is-a-document-database) written in **.NET**, offering a **flexible data model** designed to address requirements coming from real-world systems. RavenDB allows you to build high-performance, low-latency applications quickly and efficiently.

## Dapper

* <http://code.google.com/p/dapper-dot-net/>

A micro Object relational mapper, assisting in mapping database object when using an ODBC source.

# The RestClient

This library shipped with Styx helps .Net based applications to easier send requests and parse the response. Just reference the .dll file and instantiate **RestClient** to start using it.

The constructor takes one argument: *styxBaseUrl*by default Styx exposes an endpoint at [*http://localhost:1337/xml*](http://localhost:1337/xml)

## Example code

### Using the client

public void PerformSearchTest()

{

var searchRequest = new SearchRequest

{

ApiKey = "default",

Searchers = new Searcher[]

{

new FreeTextSearcher

{

Id = "hai",

MaxResults = 10,

Query = "svensson",

Skip = 0

}

}

};

var searchResponse = \_searchService.Search(searchRequest);

}

### Creating searchers

public class ExampleSearchers

{

public FacetSearcher CreateFacetSearcher(string queryText)

{

return new FacetSearcher()

{

Query = queryText,

FacetNames = new string[] {"CreateDate", "Autor"},

Id = "MainFacetSearcher" //Used for identifying the search result

};

}

public FreeTextSearcher CreateFreeTextSearcher(string queryText, int skipHits)

{

return new FreeTextSearcher()

{

Query = queryText,

MaxResults = 10,

//if each page has 10 hits and you want to view page 3, this should be set to 20

Skip = skipHits,

Id = "MainHitSearcher",

Highlight = new Highlight()

{

StartTag = "<em>",

EndTag = "</em>"

}

};

} }