

IHS > Decision Support Tool

IHS Enerdeq™ Web Services International Edition Developer's Guide

July 8, 2011 – Version 1.10 Draft 3

© 2011, IHS Inc. and its affiliated and subsidiary companies, all rights reserved. All other trademarks are the property of IHS Inc. and its affiliated and subsidiary companies.

This product, including software, data and documentation are licensed to the user for its internal business purposes only and may not be disclosed, disseminated, sold, licensed, copied, reproduced, translated or transferred to any third party.

IHS Inc.
15 Inverness Way East
Englewood, Colorado 80112
303-736-3000

Contents

Introduction	1
Target Audience.....	1
Information Available from the Services	1
Overview of Services Available	1
WSDL	2
Data Dictionary.....	3
Data Model Entity Relationship Diagram	3
Sample Data	3
Report Metadata.....	3
Java Sample Code and Sample Application.....	4
.NET Sample Code and Sample Application	5
IHS Equifax Server Certificate	6
Protocols and Standards for Web Services Interoperability	6
Compression.....	7
Note about Sample Code	7
Note about Language Parameter	7
Chapter 1 — Session	8
Consuming the Session Service	10
Refreshing a Session	11
Closing a Session.....	11
Chapter 2 — Report	12
Consuming the Report Service	15
Obtaining a Template	15
The GetReport Method	16
Chapter 3 — Graph	18
Consuming the Graph Service	21
Obtaining a Template	21
The GetGraph Method.....	22
Chapter 4 – Schema	24
Consuming the Schema Service	30
Obtaining Category Metadata	30
Obtaining Attribute Metadata	31
Obtaining Search Behavior Metadata.....	32
Chapter 5 – Named Query	33
Consuming the Named Query Service	36
Creating a Query	37
Obtaining Query Details.....	38
Removing Queries.....	39
Enerdeq Query Language Reference.....	40
Enerdeq Query Language Samples	40
Chapter 6 – Query	41
Consuming the Query Service	44
IHS_Count Queries for Information Discovery	44
Running User Defined Queries	46
Chapter 7 – Export	48

Enerdeq Web Services International Developer's Guide

Consuming the Export Service..... 51
Obtaining Details of Available Exports 52
Retrieving a Spatial Export 52
Disclaimers 56

Figures

Figure 1 - Services available from Enerdeq Web Services International 2

Figure 2 - The SessionService Interface 9

Figure 3 - The ReportService Interface 13

Figure 4 - The GraphService Interface 19

Figure 5 - The SchemaService Interface 26

Figure 6 - The NamedQueryService Interface 34

Figure 7 - The QueryService Interface 42

Figure 8 - The ExportService Interface..... 49

Introduction

Target Audience

This document is intended to be used by software developers wishing to connect to and consume the IHS Enerdeq Web Services International Edition. It is assumed that the developer has practical experience of connecting to and consuming web services with their selected development language and development environment.

For developers unfamiliar with web services get started [here](#) and [here](#). Useful resources can be found [here](#) for .NET developers while Java developers can find useful resources [here](#) and [here](#).

Prior knowledge of the IHS [EDIN](#) and [EDIN-GIS](#) reporting and graphing system is helpful but not essential when using the web services.

Information Available from the Services

Enerdeq Web Services International provides an application programming interface that allows search and retrieval of information in the [IHS International E&P Database](#). Information can be retrieved in export, report and graph formats familiar to users of EDIN and EDIN-GIS, through web service protocols.

Specifically, the following data modules are available:

[Well Data](#)

[Discoveries and Fields Data](#)

[Concessions and Contracts Data](#)

Overview of Services Available

Each of the component services supplies method calls that make it easy to implement or integrate an application to search and retrieve information in the IHS International E&P Database.

Figure 1.1 provides a summary view of the capabilities available through Enerdeq Web Services International.

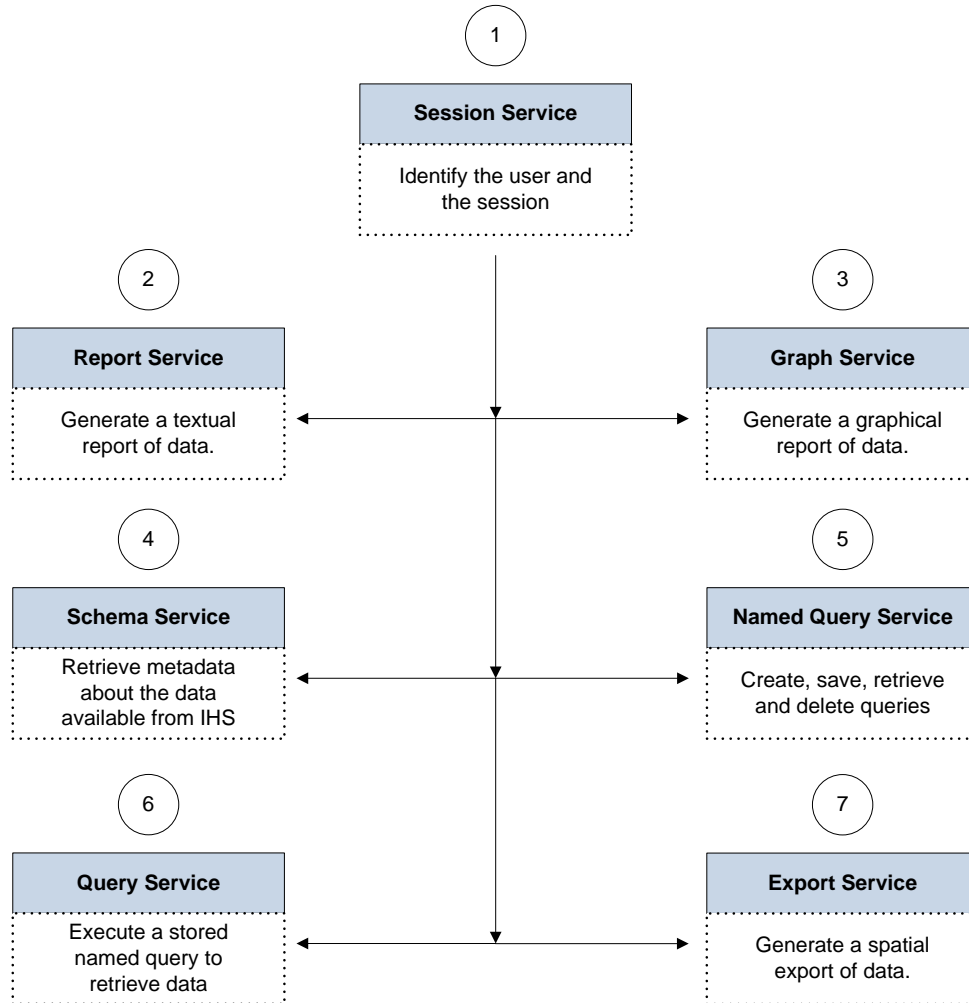


Figure 1 – Services available from Enerdeq Web Services International

WSDL

<https://services.ihs.com/Energy/International/Session.svc?WSDL>

<https://services.ihs.com/Energy/International/Schema.svc?WSDL>

<https://services.ihs.com/Energy/International/NamedQuery.svc?WSDL>

<https://services.ihs.com/Energy/International/Query.svc?WSDL>

<https://services.ihs.com/Energy/International/Report.svc?WSDL>

<https://services.ihs.com/Energy/International/Graph.svc?WSDL>

<https://services.ihs.com/Energy/International/Export.svc?WSDL>

Data Dictionary

The [Data Dictionary](#) documents the information categories and attributes available via the Schema, NamedQuery and Query Services.

Data Model Entity Relationship Diagram

The [Data Model Entity Relationship Diagram](#) documents the relationships between the information categories available via the Schema, NamedQuery and Query Services.

Sample Data

This [sample data](#) shows sample reports and graphs produced by the Report and Graph Services.

Download and extract the [zip file](#), and open the Report Graph Samples.xls file.

1	Sample								
2	Subject Area	Name	Key	Accepts multiple IDs?	TABDELIMITED	XML	EXCEL	MHTML	PDF
3	FIE	Assetbank Reports	FIEASSET	Yes	FIE_FIEASSET.txt	FIE_FIEASSET.xml	FIE_FIEASSET.xls	FIE_FIEASSET.mht	N/A
4	EPC	Block General	BLOCKGENERAL	Yes	EPC_BLOCKGENERAL.txt	EPC_BLOCKGENERAL.xml	EPC_BLOCKGENERAL.xls	EPC_BLOCKGENERAL.mht	N/A
5	EPC	Block History	BLOCKHISTORY	Yes	EPC_BLOCKHISTORY.txt	EPC_BLOCKHISTORY.xml	EPC_BLOCKHISTORY.xls	EPC_BLOCKHISTORY.mht	N/A
6	EPC	Block Outlines	BLOCKPOINTS	Yes	EPC_BLOCKPOINTS.txt	EPC_BLOCKPOINTS.xml	EPC_BLOCKPOINTS.xls	EPC_BLOCKPOINTS.mht	N/A
7	EPC	Blocks Company Interests	BLOCKCOMPINT	Yes	EPC_BLOCKCOMPINT.txt	EPC_BLOCKCOMPINT.xml	EPC_BLOCKCOMPINT.xls	EPC_BLOCKCOMPINT.mht	N/A
8	EPC	Blocks Locations	BLOCKLOC	Yes	EPC_BLOCKLOC.txt	EPC_BLOCKLOC.xml	EPC_BLOCKLOC.xls	EPC_BLOCKLOC.mht	N/A
9	EPC	Contract Block Basic	CONBLOCKBASIC	Yes	EPC_CONBLOCKBASIC.txt	EPC_CONBLOCKBASIC.xml	EPC_CONBLOCKBASIC.xls	EPC_CONBLOCKBASIC.mht	N/A
10	EPC	Contract Commitments	CONCOMS	Yes	EPC_CONCOMS.txt	EPC_CONCOMS.xml	EPC_CONCOMS.xls	EPC_CONCOMS.mht	N/A
11	EPC	Contract Company Interests	CONCOMPINT	Yes	EPC_CONCOMPINT.txt	EPC_CONCOMPINT.xml	EPC_CONCOMPINT.xls	EPC_CONCOMPINT.mht	N/A
12	EPC	Contract General	CONGENERAL	Yes	EPC_CONGENERAL.txt	EPC_CONGENERAL.xml	EPC_CONGENERAL.xls	EPC_CONGENERAL.mht	N/A
13	EPC	Contract History	CONHIST	Yes	EPC_CONHIST.txt	EPC_CONHIST.xml	EPC_CONHIST.xls	EPC_CONHIST.mht	N/A
14	EPC	Contract Locations	CONLOC	Yes	EPC_CONLOC.txt	EPC_CONLOC.xml	EPC_CONLOC.xls	EPC_CONLOC.mht	N/A
15	EPC	Contract Scheduled Events	CONSCHEVENTS	Yes	EPC_CONSCHEVENTS.txt	EPC_CONSCHEVENTS.xml	EPC_CONSCHEVENTS.xls	EPC_CONSCHEVENTS.mht	N/A
16	EPC	Contract Sheet	EPC	No	N/A	N/A	EPC.xls	EPC.mht	EPC.pdf
17	WEL	Drilling Success	WELLDRLSUC	Yes	WEL_WELLDRLSUC.txt	WEL_WELLDRLSUC.xml	WEL_WELLDRLSUC.xls	WEL_WELLDRLSUC.mht	N/A
18	FIE	Field Annual Production	FIEANPROD	Yes	FIE_FIEANPROD.txt	FIE_FIEANPROD.xml	FIE_FIEANPROD.xls	FIE_FIEANPROD.mht	N/A
19	FIE	Field Basic	FIEBASIC	Yes	FIE_FIEBASIC.txt	FIE_FIEBASIC.xml	FIE_FIEBASIC.xls	FIE_FIEBASIC.mht	N/A
20	FIE	Field Company Interests	FIECOMPINT	Yes	FIE_FIECOMPINT.txt	FIE_FIECOMPINT.xml	FIE_FIECOMPINT.xls	FIE_FIECOMPINT.mht	N/A
21	FIE	Field Costs	FIECOSTS	Yes	FIE_FIECOSTS.txt	FIE_FIECOSTS.xml	FIE_FIECOSTS.xls	FIE_FIECOSTS.mht	N/A
22	FIE	Field Cumulative Production	FIECUMPROD	Yes	FIE_FIECUMPROD.txt	FIE_FIECUMPROD.xml	FIE_FIECUMPROD.xls	FIE_FIECUMPROD.mht	N/A
23	FIE	Field Events	FIEEVENTS	Yes	FIE_FIEEVENTS.txt	FIE_FIEEVENTS.xml	FIE_FIEEVENTS.xls	FIE_FIEEVENTS.mht	N/A
24	FIE	Field General	FIEGENERAL	Yes	FIE_FIEGENERAL.txt	FIE_FIEGENERAL.xml	FIE_FIEGENERAL.xls	FIE_FIEGENERAL.mht	N/A
25	FIE	Field Image	FIEIMAGESTPC	Yes	FIE_FIEIMAGESTPC.txt	FIE_FIEIMAGESTPC.xml	FIE_FIEIMAGESTPC.xls	FIE_FIEIMAGESTPC.mht	N/A

This spreadsheet lists all the reports and graphs available from the Services, and provides hyperlinks to real samples of every report and graph in every available format.

Report Metadata

This [report metadata](#) specifies the information included in each of the reports available from the Report Service.

Download and extract the [zip file](#), and open the metadata.xls file.

Table	Column	Label	Type	Precision	Scale	Description
ADM_WELL_HEADER	COUNTRY_NAME	Country Name	VARCHAR2	50	0	The name of the country in which the well is located.
ADM_WELL_HEADER	BASIN_NAME	Basin Name	VARCHAR2	50	0	The name of the basin or sub-basin in which the well was drilled.
ADM_WELL_HEADER	WELL_NAME	Well Name	VARCHAR2	71	0	The type of name by which the well is referenced as, mainly the well name and number [Example: ...]
ADM_WELL_HEADER	ALTERNATE_WELL_NAME	Alternate Well Name	VARCHAR2	71	0	The second type of name by which the well is referenced as. [Example with a well in Denmark: ...]
ADM_WELL_HEADER	ALTERNATE_WELL_NAME_2	Alternate Well Name 2	VARCHAR2	71	0	
ADM_WELL_HEADER	WELL_ID	Well Id	NUMBER	12	0	The IRIS21 unique identification code of the well
ADM_WELL_HEADER	ONS_OFFSHORE	Onshore Offshore	VARCHAR2	20	0	The situation of the well (onshore or offshore).
ADM_WELL_HEADER	TERRAIN	Terrain	VARCHAR2	20	0	The physiographic province in which the well is located [Examples: shelf, deep water, lake, swar ...]
ADM_WELL_HEADER	LATITUDE	Latitude	VARCHAR2	20	0	The latitude of the well's surface location expressed in Greenwich degrees, minutes and seconds
ADM_WELL_HEADER	LONGITUDE	Longitude	VARCHAR2	20	0	The longitude of the well's surface location expressed in Greenwich degrees, minutes and second
ADM_WELL_HEADER	LATITUDE_DEC_DEG	Latitude Dec Deg	NUMBER	9	6	The latitude of the well's surface location expressed in Greenwich decimal degrees.
ADM_WELL_HEADER	LONGITUDE_DEC_DEG	Longitude Dec Deg	NUMBER	9	6	The longitude of the well's surface location expressed in Greenwich decimal degrees.
ADM_WELL_HEADER	COORD_QUAL	Coord Qual	VARCHAR2	20	0	The degree of accuracy of the well's surface location.
ADM_WELL_HEADER	WATER_DEPTH_METER	Water Depth Meter	NUMBER	9	2	The vertical distance expressed in metres between the sea floor and the mean sea level where th
ADM_WELL_HEADER	WATER_DEPTH_FEET	Water Depth Feet	NUMBER	9	2	The vertical distance expressed in feet between the sea floor and the mean sea level where the w
ADM_WELL_HEADER	GROUND_ELEVATION_METER	Ground Elevation Meter	NUMBER	9	2	The vertical distance expressed in metres between the mean sea level and the earth's surface wh
ADM_WELL_HEADER	GROUND_ELEVATION_FEET	Ground Elevation Feet	NUMBER	9	2	The vertical distance expressed in feet between the mean sea level and the earth's surface wher
ADM_WELL_HEADER	GROUP_NAME	Group Name	VARCHAR2	50	0	The name of the company or group of companies that drilled the well. [Examples: Total, Shell/BF ...]
ADM_WELL_HEADER	OPERATOR_NAME	Operator Name	VARCHAR2	50	0	The name of the well operator. [Examples: Amoco, Petrobras]
ADM_WELL_HEADER	CONTRACT_NAME	Contract Name	VARCHAR2	50	0	The name of the E/P contract in which the well was drilled.
ADM_WELL_HEADER	EPCSTG_ID	Epcstg Id	NUMBER	12	0	The IRIS21 unique identification code of the E/P contract stage
ADM_WELL_HEADER	BLOCK_NAME	Block Name	VARCHAR2	50	0	The name of the block in which the well was drilled.
ADM_WELL_HEADER	GA_ID	Ga Id	NUMBER	12	0	The IRIS21 unique identification code of the block
ADM_WELL_HEADER	FIELD_NAME	Field Name	VARCHAR2	50	0	The name of the field in which the well was drilled or of the discovery if the well is an outpost
ADM_WELL_HEADER	WELL_CLASS	Well Class	VARCHAR2	20	0	The classification of the well according to its purpose [Examples: New-field wildcat; Outpost; Dev ...]
ADM_WELL_HEADER	SPUD_DATE_TEXT	Spud Date Text	VARCHAR2	20	0	The date when the well was spudded. [Example: 01 Mar 1994].
ADM_WELL_HEADER	INIT_DRILL_END_DATE_TEXT	Init Drill End Date Text	VARCHAR2	20	0	The end date of the "Initial Drilling" period of operation. [Example: 01 March 1994].
ADM_WELL_HEADER	LAST_COMPLETION_DATE_TEXT	Last Completion Date Text	VARCHAR2	20	0	The end date of the last period of operation. [Example: 01 Mar 1994].

This spreadsheet includes one sheet for each report available from the Report Service. Metadata for each item on each report includes table, column, datatype, description and list of valid values. Each row also includes a hyperlink to a text file containing the list of valid values for that data item.

Java Sample Code and Sample Application

This [Java sample application](#) allows you to exercise all the available Services. The source code for this sample application is also included to assist developers building their own integration with the Services.

This application is especially useful for developers because it provides a user interface for constructing valid Enerdeq Query Language XML.

To run the application, download the [jar file](#) and then run it.

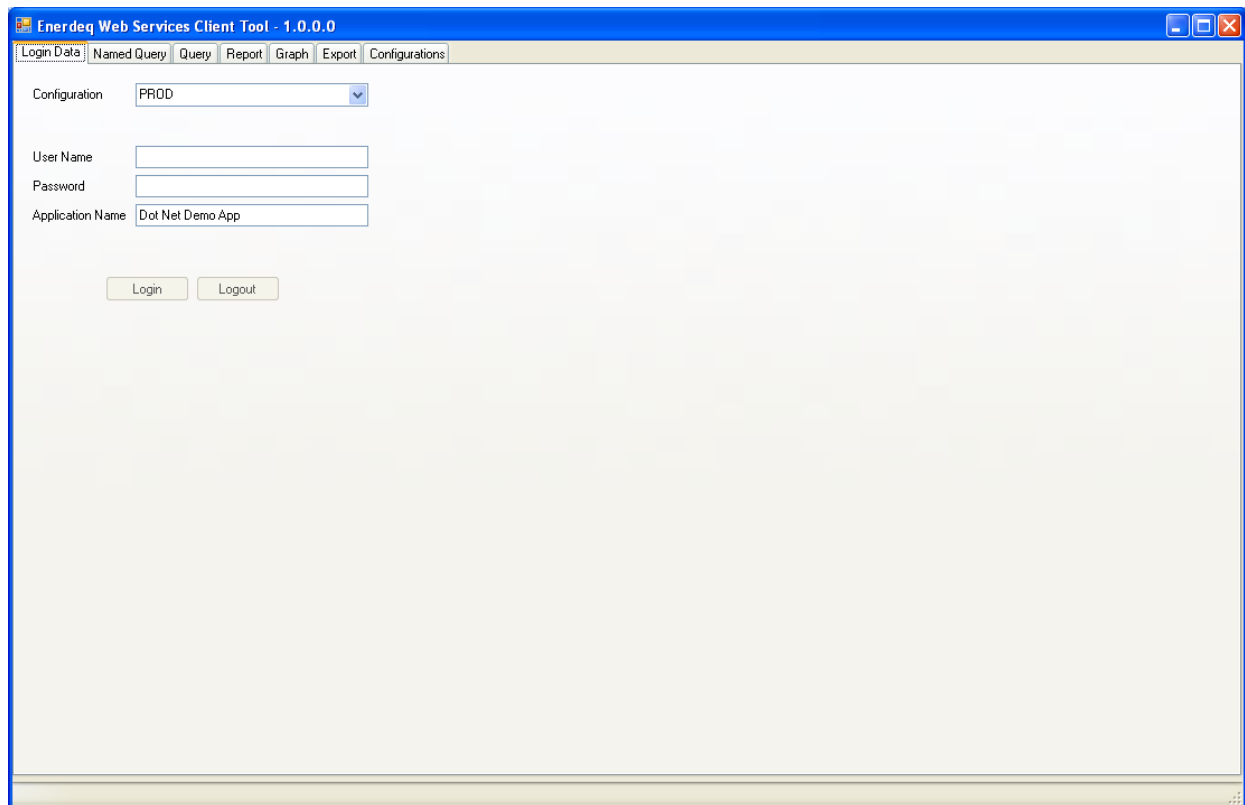


.NET Sample Code and Sample Application

This [.NET sample application](#) allows you to exercise all the available Services. The source code for this sample application is also included to assist developers building their own integration with the Services.

This application is especially useful for developers because it provides a user interface for constructing valid Enerdeq Query Language XML.

To run the application, download and extract the [zip file](#) and run WebServicesDemoUI.exe located in the ServiceTool\bin\Debug directory.



IHS Equifax Server Certificate

IHS utilizes an Equifax certificate server as part of the Web services. By having this recognized Certificate Authority issued certificate consumers can verify that the destination server belongs to and is operated by IHS. All services utilize SSL (Secure Socket Layer) to ensure integrity of all material transmitted.

Protocols and Standards for Web Services Interoperability

Enerdeq Web Services International facilitates interoperability through the following standards and protocols:

- Basic Profile 1.1
- Simple SOAP Binding 1.0
- Basic Security Profile 1.0 Working Draft
- SOAP 1.0, 1.2
- WSDL 1.1
- WS_Policy 1.2
- WS-Security 1.0
- WS-Addressing 1.0
- Web Services Interoperability

Please refer to the Simple SOAP Binding Profile (usually referred to as Basic Profile 1.1).

For specific information on the Basic Profile 1.0, see

<http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0-2004-08-24.html>.

Enerdeq Web Services International is configured to interoperate seamlessly with Silverlight and Flex clients.

Compression

Enerdeq Web Services International is configured to allow clients to request that the service responses be compressed for transmission. In order to request a compressed response the client must include the HTTP request header “Accept-Encoding: gzip, deflate” and be capable of successfully decoding the compressed response. One method of achieving this whilst using WCF 3.5 is for the client to implement a custom HTTP(S) web request module.

Note about Sample Code

For the sake of brevity the message contract is omitted from sample code and parameter sections of this document. The client endpoint configuration name is also omitted to make the sample code easier to read (line wrapping). The endpoint configuration name is configurable by service consumers but as a default will take its name from the server side definition. If the default settings are left unchanged then the naming pattern is as follows:

SOAP[version][encoding][service] e.g. SOAP12MtomReportService.

All code samples in this document are provided using .NET 3.5 C#.

Sample code is available in both [.NET C#](#) and [Java](#) languages.

Note about Language Parameter

A language parameter is used extensively throughout the service API. Language refers to the setting on the service consumer and enables globalization of the services for non-data messages i.e. information and exception messages; all data will be supplied in the language in which it is stored. Currently, however, the web services will only respond using US English, regardless of the language supplied. The parameter must be supplied in the internationalized standard ISO format of [language]-[country] e.g. en-US, en-GB. This parameter setting can be easily retrieved from the operating system of the computer being used by the consumer e.g. using the C# code `System.Globalization.CultureInfo.CurrentCulture.Name`

Chapter 1 — Session

The **SessionService** provides functionality to allow a service consumer to be screened through security and thereafter be able to access data resources. Recognizing a service consumer as a known permitted entity is commonly referred to as *authentication*. Exactly which services and what data can be accessed once a consumer is recognized are commonly referred to as *authorization*.

A session must be opened before any services can be accessed. The **Login** method is used to open a session. The pieces of information used to determine whether a service consumer is known to IHS are the name of the application calling the Services, the “user” identifier assigned by IHS and the password for that “user”.

Once the consumer is successfully identified they will be issued a session id which must then be used in all subsequent calls to the Services. Data cannot be retrieved without the consumer supplying a session id.

Sessions have a defined and finite imposed idle time - the length of time the session will remain available for use after the last activity. The period of time a session will remain available without activity is 60 minutes. A session can be kept alive manually during known periods of inactivity by using the **KeepAlive** method which will reset the idle time counter. As a security measure, all sessions will be timed out regardless of activity after 18 hours.

Even though a service will time out after the idle time has passed, a session should be explicitly closed after a consumer has completed all desired interaction with the services. This releases license locks taken at the time the session was opened and creates a clear, explicit usage audit.

Figure 2 provides a visualization of what is available via the **SessionService**, Table 1 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

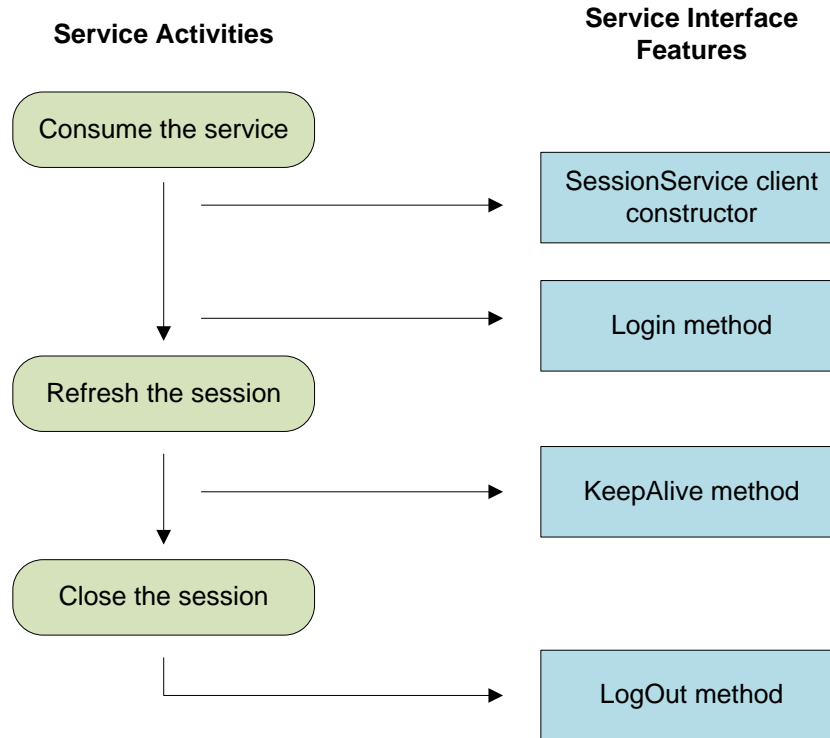


Figure 2 – The SessionService Interface

Table 1 — SessionService Interface**SessionService() + standard overloads**

Creates a session service object.

The name of the service used to establish a session. A convention is to append “Client” to the name of the service to create a proxy instance of the service:

SessionServiceClient

Login(user, password, application)

Allows a service consumer to identify themselves to IHS and open a session during which interaction with services can take place. Returns a **string** that identifies the session just opened. Accepts three mandatory arguments, all of type **string**.

User	Identifies the name of the consumer for the current session.
password	Provides the password corresponding to the username.
application	Identifies the name of the application connecting to the Services.

KeepAlive (sessionID)

Allows a service consumer to keep a session active during extended periods of inactivity. Does not have a return value. Accepts one mandatory argument, of type **string**.

sessionID Provides the identifier for the session (as returned by the **Login ()** method) to be kept alive.

LogOut (sessionID)

Closes a session of interaction. Does not have a return value. Accepts one mandatory argument, of type **string**.

sessionID Provides the identifier for the session (as returned by the **Login ()** method) to be closed.

Consuming the Session Service

To open a session, a client side proxy can be created to the **SessionService** provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the **SessionService**.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (SessionServiceClient ssc = new SessionServiceClient (...))
{
    ...
}
```

To open a session use the **SessionServiceClient.Login ()** method and supply a user name, a password and an application name. The username and password are provided by IHS while the application name is supplied by the consumer. All of the arguments are of type **string**. Below is an example of how the **Login ()** method can be used.

```
using (SessionServiceClient ssc = new SessionServiceClient (...))
{
    string username = "MyUserName";
    string password = "MyPassword";
    string applicationName = "Sample Code";
    string sessionId = string.Empty;

    // try a login
    sessionId = ssc.Login(username, password, applicationName);
}
```

As the example shows, the **Login ()** method returns a value of the **string** type, which provides the identifier for the session. This identifier should then be used in all subsequent service calls during the session.

Refreshing a Session

By default, sessions timeout after 60 minutes of inactivity. When a session times out the session id is made void and a service consumer is prevented from further access to IHS services using that identifier. If a session times out then a new session must be opened before IHS services and data can be accessed once more. If a prolonged period of inactivity is expected by a consumer then it becomes important to refresh the session in order to artificially keep it active.

To refresh a session use the `SessionServiceClient.KeepAlive()` method and supply the identifier for the session to be kept alive, the argument is a `string`. Below is an example of how the `KeepAlive()` method can be used.

```
using (SessionServiceClient ssc = new SessionServiceClient (...))
{
    ssc.KeepAlive(sessionId);
}
```

Closing a Session

Even though sessions will time out after a given idle time it is good programming practice to explicitly close them when they are no longer needed. A session can be closed at any time. When a session is closed the application will no longer be able to access services or data using the session identifier. After a session has been closed if access to services or data is required a consumer will be required to open a new session which will be characterized by a new session identifier. Below is an example of how the `SessionServiceClient.LogOut()` method can be used.

```
using (SessionServiceClient ssc = new SessionServiceClient (...))
{
    ssc.LogOut(sessionId);
}
```

Chapter 2 — Report

The **ReportService** provides mechanisms to discover details about the textual reports that are available. This includes details of available report formats and all parameters required to request a given report. The **ReportService** then permits textual reports to be retrieved in a variety of formats, such as PDF, Excel, XML and [MHTML](#).

Reports are classified against a subject area. The subject areas currently available via the web services are contracts, fields, wells and production. These are abbreviated in parameters and return values as follows:

Contracts	-	EPC
Fields	-	FIE
Wells	-	WEL
Production	-	PRO

To retrieve detailed information about the reports available use one of the **GetTemplates ()** methods which will retrieve either all the available reports or a subset of available reports restricted by subject area e.g. well related (WEL), field related (FIE), contract related (EPC) or production related (PRO). Detailed information is returned about the available reports, including all available formats, the subject area for the report and details of all possible parameters – some parameters will be mandatory while other will be optional.

To retrieve an actual report use the **GetReport ()** method. If optional parameters are not supplied then default values will be used. The report will be returned as an array of bytes, the consumer must then reform this into a “document”.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **5MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 3 provides a visualization of what is available via the **ReportService**, Table 2 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

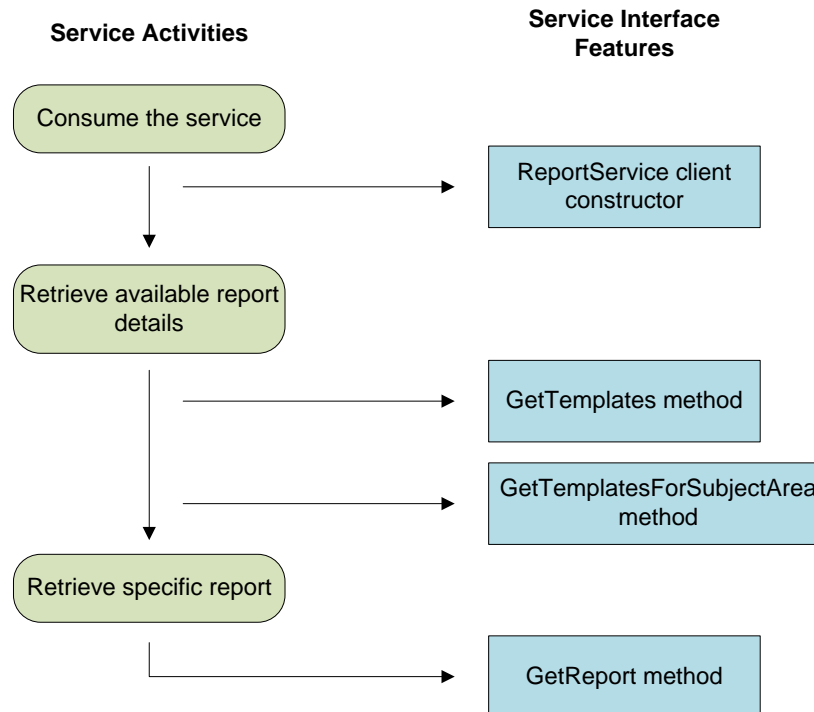


Figure 3 – The ReportService Interface

Table 2 — ReportService Interface Features**ReportService() + standard overloads**

Creates a report service object.

The name of the service used to access the report services. A convention is to append “Client” to the name of the service to create a proxy instance of the service:

ReportServiceClient.

GetTemplates(ApplicationName, SessionId, Language)

Retrieves details of available reports. Returns an array of [ReportTemplates](#). Accepts three mandatory arguments, all of type **string**.

ApplicationName Identifies the name of the application connecting to the Service.

SessionId The current session identifier (retrieved using the **SessionService**).

Language Designates the character glyph format of the client e.g. en-GB, en-US.

GetTemplatesForSubjectArea(ApplicationName, SessionId, Language, SubjectArea)

Retrieves details of available reports which has been restricted to a specified subject area e.g. wells. Returns an array of [ReportTemplates](#). Accepts four mandatory arguments, all of type **string**.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US.
SubjectArea	Restricts the report templates returned to the subject area provided e.g. FIE (field), WEL (well), EPC (contract), PRO (production).

GetReport(ApplicationName, SessionId, Language, Parameters, ReportFormat, ReportKey, SubjectArea)

Retrieves a specific report based on supplied parameters and in the requested format. Returns an array of the **bytes** representing the requested report. The consumer must translate the bytes into a “document” for human consumption. Accepts six mandatory **string** arguments and one optional **Parameters** argument.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US.
Parameters	Allows report specific parameter values to be supplied to the report engine. This is an array of the Parameter type. Details are retrieved in the ReportTemplates call. Set this argument to null if there are no specific parameters.
ReportFormat	Specifies the format for the required report. Details are retrieved in the ReportTemplates call e.g. XML.
ReportKey	Specifies the identifier for the required report. Details are retrieved in the ReportTemplates call e.g. FIEBASIC.
subjectArea	Specifies subject area the report can be found in. Details are retrieved in the ReportTemplates call e.g. FIE.

ReportTemplate (Object)

A data type used to return details about report and graph templates provided by the report and graph services. It provides detailed information about the report or graph including mandatory and optional parameters and available formats.

Description	A description of the report or graph template.
Key	The identifier (key) for the report or graph template.
Name	The name of the report or graph template.
Parameters	An array detailing the parameters supported by the report or graph. Information includes names, data type, possible values

	and whether it is mandatory or optional.
SubjectArea	Details the subject area that the report or graph belongs to e.g. FIE (field), WEL (well), EPC (contract), PRO (production).
ViableFormats	Details the formats the report or graph is available in e.g. PDF, XML, MHTML, Excel, PNG, etc.

Parameter (Object)

A data type used to return details about parameters supported by reports as part of [ReportTemplate](#) and queries as part of **GetParameters**; and for consumers to supply parameter details to a [report](#), [graph](#) or [export](#) generation request or to a query [generation](#) or query [execution](#) request.

Name	The name of the parameter.
Optional	Indicates whether the parameter is optional or mandatory.
Type	The data type of the parameter e.g. string.
Values	Indicates the possible values for the parameter supported by IHS or holds the actual value(s) being provided by the consumer.

Consuming the Report Service

To access textual reports, a client side proxy can be created to the **ReportService** provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the **ReportService**.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (ReportServiceClient rsc = new ReportServiceClient(...))
{
    ...
}
```

Obtaining a Template

To retrieve details about available report templates, either all of them or just for a specific area such as wells (WEL), fields (FIE), contracts (EPC) or production (PRO), use one of the **GetTemplates ()** methods. These methods require the name of the application being called, a valid session id and the client language; subject area must also be provided to return reports for a subject area. It returns an array of [ReportTemplates](#).

[ReportTemplate](#) describes all the details required to request a specific report along with some more human readable information such as a description of the report.

Below is an example of how the `GetTemplates()` method can be used to retrieve report details and display them on the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the return array type is represented as a generic list. If a list of report templates specific to a subject area such as wells is required, use the `GetTemplatesForSubjectArea()` method which accepts the additional `SubjectArea` parameter – shown commented out in green.

```

...
using (ReportServiceClient rsc = new ReportServiceClient(...))
{
    // read report templates
    List<ReportTemplate> templates = rsc.GetTemplates(
        applicationName,
        sessionId,
        CultureInfo.CurrentUICulture.Name);

    //List<ReportTemplate> templates = rsc.GetTemplatesForSubjectArea(
    //    applicationName,
    //    sessionId,
    //    CultureInfo.CurrentUICulture.Name,
    //    "FIE");

    foreach (ReportTemplate rt in templates)
    {
        Console.WriteLine(
            "Report template name = " + rt.Name +
            ", key = " + rt.Key +
            ", subject area = " + rt.SubjectArea +
            ", description = " + rt.Description + ".");

        foreach (Parameter p in rt.Parameters)
        {
            Console.WriteLine(
                "Parameter name = " + p.Name +
                ", optional = " + p.Optional +
                ", type = " + p.Type +
                ", values = " + p.Values + ".");
        }

        foreach (string format in rt.ViableFormats)
        {
            Console.WriteLine("Avialable format = " + format + ".");
        }
    }
}
...

```

The GetReport Method

To retrieve a report use the `GetReport()` method. This method requires details of the report required and returns an array of `bytes`. The byte array provides data which must be translated into a suitable format for human consumption by the service consumer e.g. file saved to hard disk. The method takes six mandatory arguments and one optional argument; these arguments

specify the application name, a valid session id, client language, key identifier, format and subject area for the report along with possible parameters.

The `GetReport()` method returns bytes representing the requested report in the requested format. The formats are defined per report in the report template. Below is an example of how to use the `GetReport()` method and write the returned xml report to the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the parameter array type is represented as a generic list.

```
...
using (ReportServiceClient rsc = new ReportServiceClient(...))
{
    List<Parameter> parameters = new List<Parameter>()
    {
        new Parameter()
        {
            Name = "ids",
            Optional = false,
            Type = "String",
            Values = "100000002195"
        }
    };

    // read xml report format
    byte[] results = rsc.GetReport(
        applicationName,
        sessionId,
        CultureInfo.CurrentCulture.Name,
        parameters,
        "XML",
        "FIEBASIC",
        "FIE");

    using (StreamReader sr = new StreamReader(new MemoryStream(results)))
    {
        Console.WriteLine("Report xml: " + sr.ReadToEnd());
    }
}
...
```

There is a limit to how much information can be passed in to the service in a single string (used for supplying the ids used to generate a report). This limit is 131072 characters which equates to approximately 10,000 ids.

Chapter 3 — Graph

The **GraphService** provides access to a number of graphs such as production graphs and creaming curves. This service provides methods to enable the consumer to discover which graphs types are available, and what parameters are required to request each type of graph. Graphs are provided in an image format such as PNG.

Graphs are classified against a subject area. The subject areas currently available via the **GraphService** are contracts, fields and wells. These are abbreviated in parameters and return values as follows:

Contracts	-	EPC
Fields	-	FIE
Wells	-	WEL

To retrieve detailed information about the graphs available use one of the **GetTemplates()** methods which will retrieve either all the available graphs or a subset of available graphs restricted by subject area e.g. well related (WEL), field related (FIE) or contract related (EPC). Detailed information is returned about the available graphs, including all available formats, the subject area for the report and details of all possible parameters – some parameters will be mandatory while other will be optional.

To retrieve an actual graph use the **GetGraph()** method. If optional parameters are not supplied then default values will be used. The graph will be returned as an array of bytes, the consumer must then reform this into a “document”.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **2MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 4 provides a visualization of what is available via the **GraphService**, Table 3 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

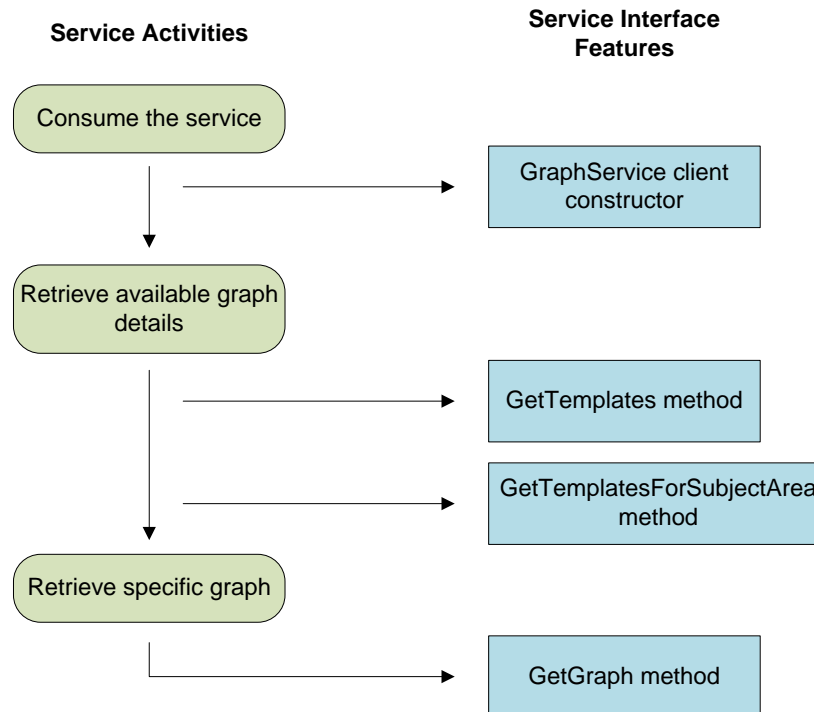


Figure 4 – The GraphService Interface

Table 3 — GraphService Interface Features**GraphService() + standard overloads**

Creates a graph service object.

The name of the service used to access the graph services. A convention is to append “Client” to the name of the service to create a proxy instance of the service:

GraphServiceClient.

GetTemplates(ApplicationName, SessionId, Language)

Retrieves details of available graphs. Returns an array of [ReportTemplates](#). Accepts three mandatory arguments, all of type **string**.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US.

GetTemplatesForSubjectArea(ApplicationName, SessionId, Language, SubjectArea)

Retrieves details of available graphs which has been restricted to a specified subject area e.g. wells. Returns an array of [ReportTemplates](#). Accepts four mandatory arguments, all of type **string**.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US.
SubjectArea	Restricts the graph templates returned to the subject area provided e.g. FIE (field), WEL (well), EPC (contract).

GetGraph(ApplicationName, SessionId, Language, Parameters, ReportFormat, ReportKey, SubjectArea)

Retrieves a specific graph based on supplied parameters and in the requested format. Returns an array of the **bytes** representing the requested graph. The consumer must translate the bytes into a “document” for human consumption. Accepts six mandatory **string** arguments and one optional **Parameters** argument.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US.
Parameters	Allows report specific parameter values to be supplied to the graph engine. This is an array of the Parameter type. Details are retrieved in the ReportTemplates call. Set this argument to null if there are no specific parameters.
ReportFormat	Specifies the format for the required graph. Details are retrieved in the ReportTemplates call e.g. PNG.
ReportKey	Specifies the identifier for the required graph. Valid values can be retrieved using a ReportTemplates call e.g. FIECOMPANIES.
subjectArea	Specifies subject area the graph can be found in. Valid values can be retrieved using a ReportTemplates call e.g. FIE.

ReportTemplate (Object)

See [details](#) shown in report service description.

Parameter (Object)

See [details](#) shown in report service description.

Consuming the Graph Service

To access graphs, a client side proxy can be created to the **GraphService** provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the **GraphService**.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (GraphServiceClient gsc = new GraphServiceClient(...))
{
    ...
}
```

Obtaining a Template

To retrieve details about available graph templates, either all of them or just for a specific area such as wells (WEL), fields (FIE) or contracts (EPC), use one of the **GetTemplates()** methods. These methods require the name of the application being called, a valid session id and the client language; subject area must also be provided to return graphs for a subject area. They return an array of [ReportTemplates](#).

[ReportTemplate](#) describes all the details required to request a specific graph along with some more human readable information such as a description of the graph.

Below is an example of how the **GetTemplates()** method can be used to retrieve graph details and display them on the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the return array type is represented as a generic list. If a list of graph templates specific to a subject area such as wells is required, use the **GetTemplatesForSubjectArea()** method which accepts the additional **SubjectArea** parameter – shown commented out in green.

```
...
using (GraphServiceClient gsc = new GraphServiceClient(...))
{
    // read report templates
    List<ReportTemplate> templates = gsc.GetTemplates(
        applicationName,
        sessionId,
        CultureInfo.CurrentCulture.Name);

    //List<ReportTemplate> templates = gsc.GetTemplatesForSubjectArea(
    //    applicationName,
    //    sessionId,
    //    CultureInfo.CurrentCulture.Name,
    //    "FIE");
}
```

```

foreach (ReportTemplate rt in templates)
{
    Console.WriteLine(
        "Graph template name = " + rt.Name +
        ", key = " + rt.Key +
        ", subject area = " + rt.SubjectArea +
        ", description = " + rt.Description + ".");

    foreach (Parameter p in rt.Parameters)
    {
        Console.WriteLine(
            "Parameter name = " + p.Name +
            ", optional = " + p.Optional +
            ", type = " + p.Type +
            ", values = " + p.Values + ".");
    }

    foreach (string format in rt.ViableFormats)
    {
        Console.WriteLine("Available format = " + format + ".");
    }
}
...

```

The GetGraph Method

To retrieve a graph use the `GetGraph()` method. This method requires details of the graph required and returns an array of `bytes`. The byte array provides data which must be translated into a suitable format for human consumption by the service consumer e.g. file saved to hard disk. The method takes six mandatory arguments and one optional argument; these arguments specify the application name, a valid session id, client language, key identifier, format and subject area for the graph along with possible parameters.

The `GetGraph()` method returns bytes representing the requested report in the requested format. The formats are defined per graph in the graph (report) template. Below is an example of how to use the `GetGraph()` method and write the returned Portable Network Graphic to the users temporary directory. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the parameter array type is represented as a generic list.

```

...
using (GraphServiceClient gsc = new GraphServiceClient(...))
{
    List<Parameter> parameters = new List<Parameter>()
    {
        new Parameter()
        {
            Name = "id",
            Optional = false,
            Type = "String",
            Values = "100000002195"
        }
    };
}

```

```
// read xml report format
byte[] results = gsc.GetGraph(
    applicationName,
    sessionId,
    CultureInfo.CurrentUICulture.Name,
    parameters,
    "PNG",
    "FIECOMPANIES",
    "FIE");

string appDataPath = Environment.GetFolderPath(
    Environment.SpecialFolder.ApplicationData);

using (FileStream fs = new FileStream(
    Path.Combine(appDataPath, "Graph.png"),
    FileMode.OpenOrCreate)
{
    fs.Write(results, 0, results.Length);
}
}
...
```

There is a limit to how much information can be passed in to the service in a single string (used for supplying the ids used to generate a graph). This limit is 131072 characters which equates to approximately 10,000 ids.

Chapter 4 – Schema

The **SchemaService** provides mechanisms for the service consumer to discover details about the information available via the Named Query and Query services. This includes full details of available [categories](#) along with associated [attributes](#) and [search behaviors](#). This service is intended to be used as a primer to utilizing the query services i.e. all information required to build queries should be available via the Schema Service.

The schema provided by Enerdeq Web Services International is organized into Category Sets, Categories, Views and Attributes. Each of Category Sets, Categories and Views can also have Search Behaviors associated with them for performing full text searches. Collectively all these items are known as entities, definitions of these entities are below:

Category Set	Group of related Categories. In the International schema each Category will belong to only one Category Set and each Category Set will contain a single Category. Category Set names have the suffix “_DOMAIN”.
Category	A subject area of interest; synonymous with a database table. NOTE: Category names will differ from report / graph subject area names since schema services (schema, named query & query) interrogate raw data while reporting services (graph & report) provide processed data.
View	Synonymous with a database view. There are currently no Views in the International schema.
Attribute	A property of a subject area of interest; synonymous with a database table column.
Search Behavior	Settings associated with performing a full text search describing which Attribute will be used during the search.

The details discovered through the **SchemaService** include both the friendly name for an entity and its web services system key. The system key information is required by the methods used to create, store and execute queries via the [NamedQueryService](#) and [QueryService](#).

Since within Enerdeq Web Services International there are no Views and there is a one-to-one relationship between CategorySets and Categories this documentation is largely going to ignore CategorySets and Views while concentrating on Categories, Attributes and SearchBehaviors.

To retrieve information about the tables exposed through the query services use the **GetCategories()** method. To retrieve details about the columns on those tables and related tables use one of the **Get..Attributes()** methods. To retrieve full text search behaviors use the **GetSearchBehaviors()** method. To retrieve the web service defined data types use the **GetAttributeTypes()** method.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **2MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 5 provides a visualization of what is available via the **SchemaService**, Table 4 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

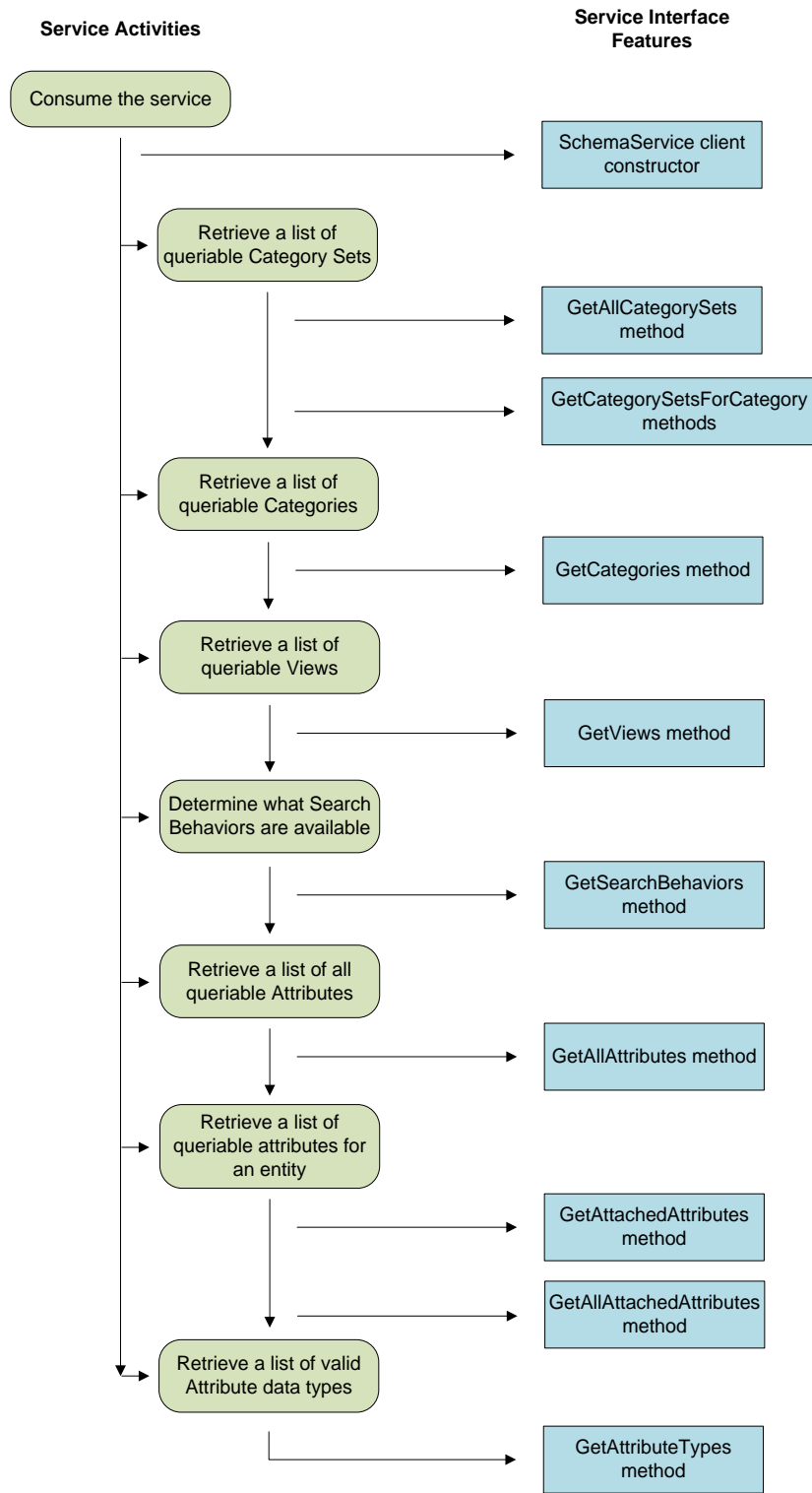


Figure 5 – The SchemaService Interface

Table 4 — SchemaService Interface**SchemaService() + standard overloads**

Creates a schema service object.

The name of the service used to retrieve details about exposed data items. A convention is to append “Client” to the name of the service to create a proxy instance of the service: **SchemaServiceClient**

GetAllCategorySets(ApplicationName, SessionId, Language)

Retrieves a list of all available category sets. Each category set is returned as an [Entity](#) object. Accepts three mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetCategorySetsForCategory(ApplicationName, SessionId, EntityKey, Language)

Retrieves a list of category sets for a specified category, in the ADM schema this is a one-to-one relationship. Each category set is returned as an [Entity](#) object. Accepts four mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
EntityKey	The system key for the category to return the category sets for. Category keys can be retrieved using the GetCategories(..) method.
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetCategories(ApplicationName, SessionId, Language)

Retrieves a list of all available categories. Each category is returned as an [Entity](#) object. Categories are required when creating queries, see the section [Named Query](#). Accepts three mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetViews(ApplicationName, SessionId, Language)

Retrieves a list of all available views. Each view is returned as an [Entity](#) object. Accepts three mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetSearchBehaviors(ApplicationName, SessionId, EntityKey, Language)

Retrieves a list of search behaviors for a specified entity. Each search behavior is returned as an [Entity](#) object. Search behaviors are required when creating full text search queries, see the section [Named Query](#). Accepts four mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
EntityKey	The system key for the entity to return the search behaviors for.
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetAllAttributes(ApplicationName, SessionId, Language)

Retrieves a list of all attributes available from all entities. Each attribute is returned as an [AttributeInfo](#) object. Accepts three mandatory **string** arguments. **NOTE:** Attributes retrieved using this method **are not** associated with an entity.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetAttachedAttributes(ApplicationName, SessionId, EntityKey, FullDepth, Language)

Retrieves a list of attributes available from the specified entity. Each attribute is returned as an [AttributeInfo](#) object. This method will include all attributes from related entities. Accepts four mandatory **string** arguments and one mandatory **bool** argument.

ApplicationName	Identifies the name of the application connecting to the
------------------------	--

	Service.
SessionId	The current session identifier (retrieved using the SessionService).
EntityKey	The system key for the entity to return the attributes for.
FullDepth	A Boolean indicator to show whether a full depth retrieval of attached attributes should be carried out or not.
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetAllAttachedAttributes (ApplicationName, SessionId, EntityKey, Language)

Retrieves a list of all attributes available from the specified entity. Each attribute is returned as an [AttributeInfo](#) object. This method will include all attributes from related entities. Equivalent to a **GetAttachedAttributes** (..) call made with “full depth” set to true. Accepts four mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
EntityKey	The system key for the category to return the category sets for. Category keys can be retrieved using the GetCategories (..) method.
Language	Designates the character glyph format of the client e.g. en-GB, en-US

GetAttributeTypes (ApplicationName, SessionId)

Retrieves a list of all the attribute type names supported by the web services. Accepts two mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).

Entity (Object)

A data type used to return details about a system entity: category set, category, search behavior, view.

Key	The key used within the web services to identify the entity.
Name	The database name of the entity.

AttributeInfo (Object)

A data type used to return details about a system attribute.

Attributes	Contains a list of nested attributes, for composite data types.
AttributeType	The web service defined type of the attribute.

Entity	The system key of the entity the attribute belongs to. This may be empty indicating that the attribute belongs to the current entity i.e. it is “local”.
Key	The key used within the web services to identify the attribute.
MultiValue	A Boolean indicator of whether the attribute can hold more than one value at once.
Name	The database name of the attribute.
ReturnType	The .NET data type.
Via	The relationship used to retrieve the attribute. This is empty for attributes which belong to the current entity.

Consuming the Schema Service

To access details of the International schema, a client side proxy can be created to the **SchemaService** provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the **SchemaService**.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (SchemaServiceClient ssc = new SchemaServiceClient(...))
{
    ...
}
```

Obtaining Category Metadata

To retrieve details about the categories available for use within the query system use the **GetCategories()** method. This method requires the name of the application being called, a valid session id and the client language. An array of [Entity](#)s is returned. An [Entity](#) describes the key and name of each category.

Below is an example of how the **GetCategories()** method can be used to retrieve details of categories and display them on the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the return array type is represented as a generic list.

```
...
using (SchemaServiceClient ssc = new SchemaServiceClient(...))
{
    // read details of exposed categories
    List<Entity> categories = ssc.GetCategories(
        applicationName,
        sessionId,
        CultureInfo.CurrentCulture.Name);
}
```

```

foreach (Entity e in categories)
{
    Console.WriteLine(
        "Category Key = " + e.Key +
        ", name = " + e.Name + ".");
}
}
...

```

Obtaining Attribute Metadata

To retrieve details about the attributes available for use within the query system along with how to navigate to them use one of the `Get..Attributes()` methods. These methods allow a consumer to retrieve details of attributes (aka table columns) available via the web services. Attributes can be retrieved from several levels: all attributes for the entire system, all attributes for an entity or attributes for a depth (deep or shallow) for an entity. Retrieving attributes for an entity at `FullDepth` will retrieve attributes for all related entities also.

Attributes retrieved using the `Get..Attributes()` methods can then be used when building queries. All the details required during query building are discoverable when retrieving metadata for attributes. The key details are: **Key**, **Entity** and **Via** as these describe exactly how to retrieve data for an attribute.

Below is an example of how the `Get.Attributes()` methods can be used to retrieve details of attributes and display them on the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the return array type is represented as a generic list.

```

...
using (SchemaServiceClient ssc = new SchemaServiceClient(...))
{
    // read details of all available attributes
    List<AttributeInfo> attributes = ssc.GetAllAttributes(
        applicationName,
        sessionId,
        CultureInfo.CurrentCulture.Name);

    OutputAttributes(attributes);

    // read details of all attributes attached to the category CONTRACT_HEADER
    attributes = ssc.GetAllAttachedAttributes(
        applicationName,
        sessionId,
        "DBO_CONTRACT_HEADER",
        CultureInfo.CurrentCulture.Name);

    OutputAttributes(attributes);

    // read top level details of attributes attached to the category
    // CONTRACT_HEADER
    attributes = ssc.GetAttachedAttributes(
        applicationName,
        sessionId,

```

```

        "DBO_CONTRACT_HEADER",
        false,
        CultureInfo.CurrentCulture.Name);

    OutputAttributes(attributes);
}
...
/// <summary>
/// Lists attribute properties to the console. Recursive.
/// </summary>
/// <param name="attributes">The attributes to list
/// details for.</param>
private void OutputAttributes(List<AttributeInfo> attributes)
{
    foreach (AttributeInfo a in attributes)
    {
        Console.WriteLine(
            "Attribute type = " + a.AttributeType +
            ", owning entity = " + a.Entity +
            ", key = " + a.Key +
            ", is multi-value = " + a.MultiValue +
            ", name = " + a.Name +
            ", return type = " + a.ReturnType +
            ", reached via = " + a.Via + ".");

        // recurse if necessary
        if (a.Attributes.Count > 0)
        {
            OutputAttributes(a.Attributes);
        }
    }
}
...

```

Obtaining Search Behavior Metadata

To retrieve details about the search behaviors available to perform full text searches across entities use the `GetSearchBehaviors()` method. This method allows a service consumer to retrieve the friendly name and system key for the search behaviors defined on a specified entity.

```

...
using (SchemaServiceClient ssc = new SchemaServiceClient(...))
{
    // read details of all search behaviors for the category CONTRACT_HEADER
    List<Entity> behaviors = ssc.GetSearchBehaviors(
        applicationName,
        sessionId,
        "DBO_CONTRACT_HEADER",
        CultureInfo.CurrentCulture.Name);

    foreach (Entity e in behaviors)
    {
        Console.WriteLine(
            "Category Key = " + e.Key +
            ", name = " + e.Name + ".");
    }
}
...

```

Chapter 5 – Named Query

The **NamedQueryService** provides functionality to allow a service consumer to create, retrieve, update and delete xml structured data queries for use in interactive querying via the **QueryService**.

All queries are constructed using Enerdeq Query Language and supplied to the service as an xml string. A separate document is provided which details the [Enerdeq Query Language](#) structure and use of each element in the construction of a query.

An [Xml Schema Document \(XSD\)](#) is available to validate the Enerdeq Query Language to be submitted to the **NamedQueryService**.

To retrieve the names of the queries already stored in the system use the **GetNamedQueries ()** method. To retrieve details of a specific query use the **GetNamedQueryDetail ()** method. To retrieve the parameters for a specific query use the **GetParameters ()** method.

To save a query use the **Register ()** method. To remove a query use the **Remove ()** method. To remove all user defined queries for the logged in user use the **RemoveAll ()** method.

Queries are registered with the system based on the query name supplied and the user logged in at the time of registration. A user does not have permission to access the queries registered by another user. It is important to note that if a user registers a query with the same name as an existing query then an update will be performed, no warning will be issued.

Several pre-defined queries have been registered by IHS with the system; these are available to all users. All these queries are prefixed with “IHS_”. The IHS queries cannot be altered or overwritten by users; they may however be “hidden” by user specific queries that are given the same name. Several queries are prefixed with “IHS_Count”, these can only be used in the [Query Service Count](#) operation. Their purpose is to support discovery of information outside of user entitlements without providing the actual data. E.g. a user who is only entitled to view data for UK wells is able to retrieve a count of wells in Norway. Example code for calling these un-entitled count queries can be found in section [Running IHS Defined Queries](#).

Avoid naming user queries with an IHS_ prefix to prevent potential conflicts.

The code samples shown in this section use message contracts which have not previously been demonstrated in code samples in order to keep the samples short and simple. The **GetQueryDetail ()** method returns more than one value and therefore the code is simplified by use of the message contracts. Without the message contracts multiple “out” parameters would be required in addition to a return value. Generally, a client can decide at the time of generating a proxy to a service whether message contracts will be used by default or not.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **2MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 6 provides a visualization of what is available via the **NamedQueryService**, Table 5 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

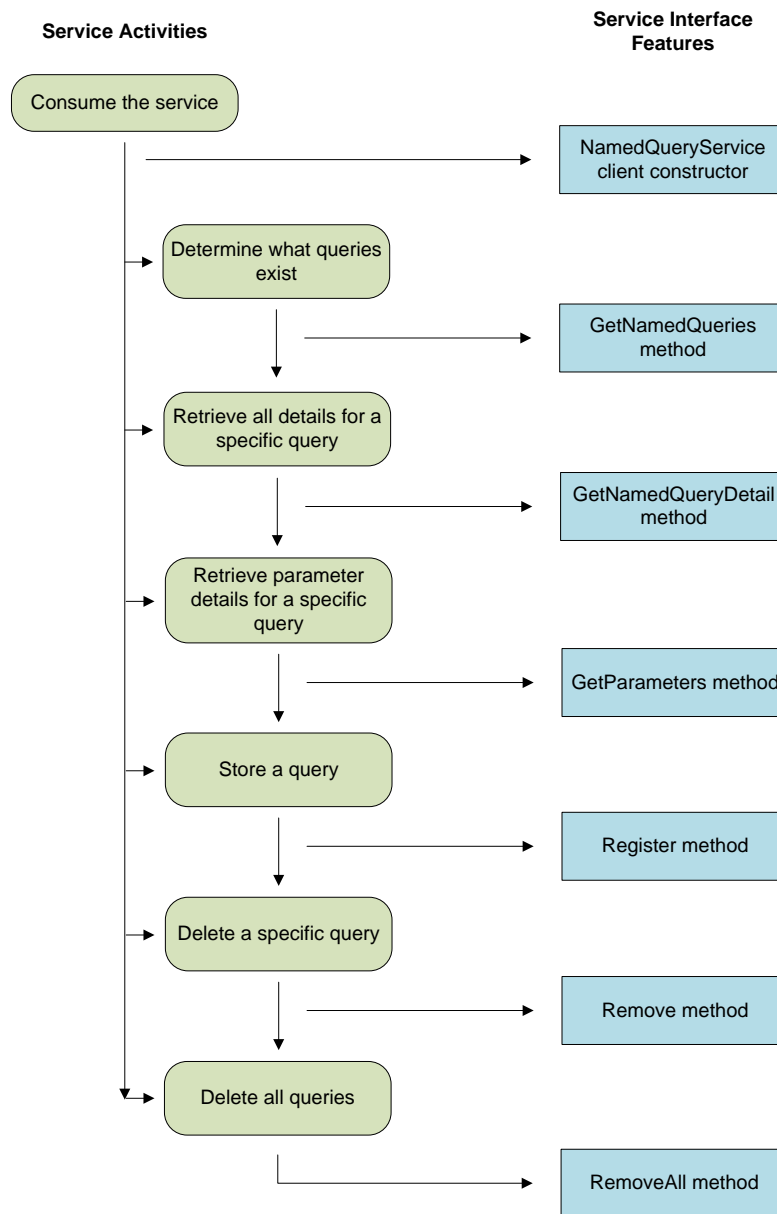


Figure 6 – The NamedQueryService Interface

Table 5 — NamedQueryService Interface**NamedQueryService() + standard overloads**

Creates a named query service object.

The name of the service used to maintain stored data queries. A convention is to append “Client” to the name of the service to create a proxy instance of the service:

NamedQueryServiceClient

GetNamedQueries(ApplicationName, SessionId)

Retrieves a list of the names of all of the queries registered on the system for the logged in user. Accepts two mandatory **string** arguments.

ApplicationName Identifies the name of the application connecting to the Service.

SessionId The current session identifier (retrieved using the **SessionService**).

GetNamedQueryDetail(ApplicationName, SessionId, NamedQuery)

Retrieves details of the specified query. These details include the name given to the query, the description given to the query and the full Enerdeq Query Language xml string as it was supplied at query registration. Accepts three mandatory **string** arguments.

ApplicationName Identifies the name of the application connecting to the Service.

SessionId The current session identifier (retrieved using the **SessionService**).

NamedQuery The name of the query to be retrieved.

GetParameters(ApplicationName, SessionId, NamedQuery)

Retrieves a list of [Parameters](#) for a specified query. Accepts three mandatory **string** arguments.

ApplicationName Identifies the name of the application connecting to the Service.

SessionId The current session identifier (retrieved using the **SessionService**).

NamedQuery The name of the query to be retrieved.

Register(ApplicationName, SessionId, Description, NamedQuery, Query)

Registers a query with the supplied details into the system for the logged in user. If a query with the same name already exists it *will* be overwritten without issuing a warning. Accepts five mandatory **string** arguments.

ApplicationName Identifies the name of the application connecting to the

	Service.
SessionId	The current session identifier (retrieved using the SessionService).
Description	The description of the query being registered.
NamedQuery	The name given to the query being registered.
Query	The query to be registered - represented as an Enerdeq Query Language xml string.

Remove(ApplicationName, SessionId, NamedQuery)

Removes a specified registered query from the system for the logged in user. Accepts three mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
NamedQuery	The name of the query to be removed.

RemoveAll(ApplicationName, SessionId)

Removes all registered queries from the system for the logged in user. Accepts two mandatory **string** arguments.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).

Parameter (Object)

See [details](#) shown in report service description.

Consuming the Named Query Service

To administer named queries, a client side proxy can be created to the **NamedQueryService** provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the **NamedQueryService**.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (NamedQueryServiceClient nqsc = new NamedQueryServiceClient (...))
{
    ...
}
```

Creating a Query

To create a new query or update an existing query on the system for the currently logged in user use the `Register()` method. This method requires the name of the application being called, a valid session id, a name for the query, a description for the query and the Enerdeq Query Language xml string itself. This method does not return anything.

Below is an example of how to use the `Register()` method to store the following SQL select statement:

```
SELECT FIE_ID, FIELD_NAME, BASIN_NAME, COORD_LOCATION,
       COUNTRY_NAMES, CUR_GROUP_NAMES, CUR_OPERATOR_NAMES,
       DATE_OF_LAST_DB_UPDATE, DATE_OF_LAST_REFRESH, DATE_OF_ENTRY
FROM   FIELD_HEADER
WHERE  (COUNTRY_NAMES LIKE '%Cuba%')
ORDER BY FIELD_NAME
```

For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions.

```
...
using (NamedQueryServiceClient nqsc = new NamedQueryServiceClient (...))
{
    string queryString =
        "<?xml version='1.0' encoding='utf-8'?>" +
        "<Query xmlns='services.ihs.com/Search'>" +
        "  <Attributes>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>FIE_ID_N</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>FIELD_NAME_S</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>BASIN_NAME_S</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>COORD_LOCATION_S</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>COUNTRY_NAMES_LS</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>CUR_GROUP_NAMES_LS</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>CUR_OPERATOR_NAMES_LS</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>DATE_OF_LAST_DB_UPDATE_DT</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>DATE_OF_LAST_REFRESH_DT</Attribute>" +
        "    <Attribute Entity='DBO_FIELD_HEADER'>DATE_OF_ENTRY_DT</Attribute>" +
        "  </Attributes>" +
        "  <Source>" +
        "    <Categories Categories='DBO_FIELD_HEADER' />" +
        "  </Source>" +
        "  <Constraints>" +
        "    <Wildcard>" +
        "      <Attribute Entity='DBO_FIELD_HEADER'>COUNTRY_NAMES_LS</Attribute>" +
        "      <Value>" +
        "        <Parameter Name='CountryName' />" +
        "      </Value>" +
        "    </Wildcard>" +
        "  </Constraints>" +
        "  <Sort>" +
```

```

        "<SortOrder Entity='DBO_FIELD_HEADER'
        Direction='Ascending'>FIELD_NAME_S</SortOrder>" +
        "</Sort>" +
        "</Query>";

        // save the query
        RegisterMessage query = new RegisterMessage(
            applicationName,
            sessionId,
            "A test query for retrieving field details by country name.",
            "IHS_TestQuery",
            queryString);

        nqsc.Register(query);
    }
    ...

```

The [Java](#) and [.NET](#) sample applications are useful tools to help developers construct and test Enerdeq Query Language. See the NamedQuery and Query tabs on these sample applications.

There is a limit to how much information can be passed in to the service in a single string (used for supplying the actual query text). This limit is 131072 characters.

Obtaining Query Details

To retrieve a list of the queries registered with the system for the logged in user use the `GetQueries()` method. This method requires the name of the application being called and a valid session id, it will return a list of query names which can then be used to find out further details about a specific query.

To retrieve the actual Enerdeq Query Language xml string that will be used if the query is executed use the `GetQueryDetail()` method. This method requires the name of the application being called, a valid session id and the name of the query to be retrieved. The full query xml string along with the description the user set for the query will be retrieved.

To retrieve the parameter settings required to execute the query use the `GetParameters()` method. This method requires the name of the application being called, a valid session id and the name of the query to be retrieved. A list of parameters is returned, this can then be used to execute a named query by supplying values for the mandatory parameters.

Below is an example of how to use the methods explained above and write the returned information to the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions.

```

...
using (NamedQueryServiceClient nqsc = new NamedQueryServiceClient (...))
{
    // see what queries we have to start with
    GetNamedQueriesResponse queries = nqsc.GetNamedQueries(
        new GetNamedQueriesMessage(applicationName, sessionId));
}

```

```

foreach (string name in queries.Names)
{
    Console.WriteLine("Query name = " + name + ".");
}

// retrieve query details, this code assumes there is a query called
// IHS_TestQuery (created in "Creating a Query" sample above)
GetNamedQueryDetailResponse details = nqsc.GetNamedQueryDetail(
    new GetNamedQueryDetailMessage(
        applicationName,
        sessionId,
        "IHS_TestQuery"));

Console.WriteLine("Query name = " + details.NamedQuery + ".");
Console.WriteLine("Query description = " + details.Description + ".");
Console.WriteLine("Query xml string = " + details.Query + ".");

// check the parameters
GetParametersResponse parameters = nqsc.GetParameters(
    new GetParametersMessage(
        applicationName,
        sessionId,
        "IHS_TestQuery"));

foreach (Parameter p in parameters.Parameters)
{
    Console.WriteLine(
        "Parameter name = " + p.Name +
        ", optional = " + p.Optional +
        ", type = " + p.Type +
        ", values = " + p.Values + ".");
}
...

```

Removing Queries

To remove a specific query from the system for the currently logged in user use the **Remove()** method. This method requires the name of the application being called, a valid session id and the name of the query to be removed. This method does not return anything.

To remove *all* queries from the system for the currently logged in user use the **RemoveAll()** method. This method requires the name of the application being called and a valid session id. This method does not return anything.

Below is an example of how to use the **Remove()** and **RemoveAll()** methods. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions.

```

...
using (NamedQueryServiceClient nqsc = new NamedQueryServiceClient (...))
{
    // delete the query
    nqsc.Remove(
        new RemoveMessage(applicationName, sessionId, "IHS_TestQuery"));

    // delete all the queries
}

```

```
nqsc.RemoveAll (  
    new RemoveMessage (applicationName, sessionId) );  
}  
...
```

Enerdeq Query Language Reference

This [manual](#) documents the EQL structure and how to construct valid EQL.

The [Java](#) and [.NET](#) sample applications are a useful tools to help developers construct and test EQL. See the NamedQuery and Query tabs on these sample applications.

Enerdeq Query Language Samples

This [document](#) provides many sample queries that will be useful to help developers construct valid EQL.

Chapter 6 – Query

The **QueryService** provides the ability to execute queries registered via the [NamedQueryService](#). All queries to be executed must be registered with this system prior to execution; ad-hoc queries are not supported. All counts and results are based on **entitled** data which may be equivalent to or less than the total data IHS holds for a particular area of interest.

To determine how many records match a query without actually retrieving the data itself, use the **Count ()** method. To retrieve a count of records based on the unique values of the selected attributes within the query (i.e. group-by count) use the **Tally ()** method.

To retrieve the actual data records which match a query use the **Query ()** method. Data is returned as an array of **bytes** conforming to the [structured content schema](#). The consumer must then reform this into an xml string in order to perform application specific business processing on it for end use. The xml string will conform to the Structured Content Schema.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **5MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 7 provides a visualization of what is available via the `QueryService`, Table 6 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

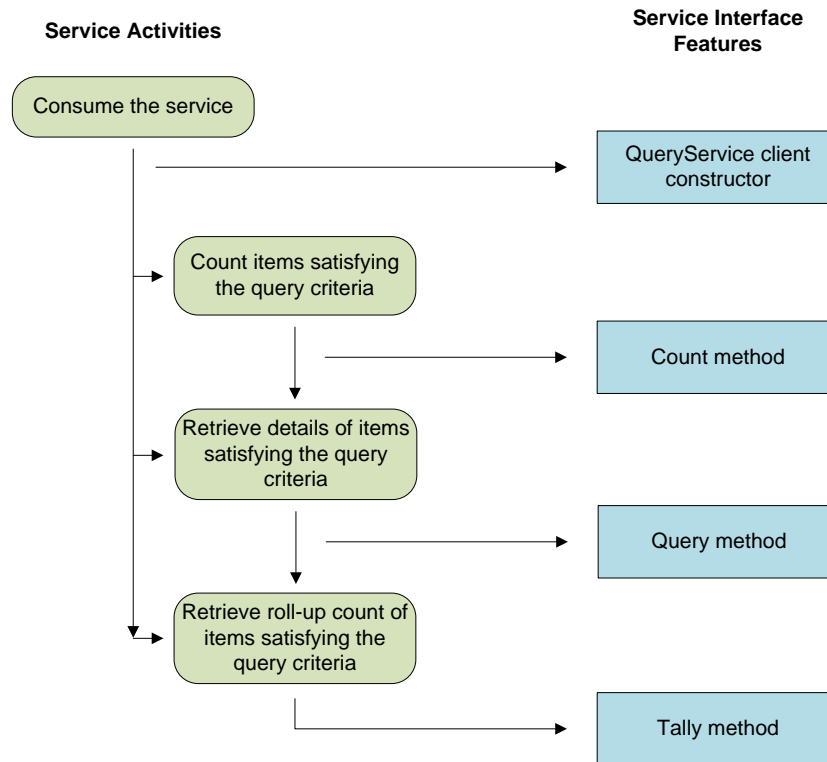


Figure 7 – The QueryService Interface

Table 6 — QueryService Interface Features

QueryService() + standard overloads

Creates a query service object.

The name of the service used to execute stored queries. A convention is to append “Client” to the name of the service to create a proxy instance of the service:

QueryServiceClient.

Count(ApplicationName, SessionId, NamedQuery, Parameters)

Retrieves a total count of results the currently logged in user is entitled to for the specified named query, which must already be stored in the system. Returns an `int`. Accepts three mandatory `string` arguments and one optional `Parameters` argument.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the <code>SessionService</code>).
NamedQuery	The name of the registered query required to be executed.
Parameters	Allows query specific parameter values to be supplied to the

query engine. This is an array of the [Parameter](#) type. Details are retrieved using the `NamedQueryService` [GetParameters\(\)](#) method. Set this argument to `null` if there are no specific parameters.

Query(ApplicationName, SessionId, NamedQuery, Page, PageSize, Parameters)

Retrieves the data results the currently logged in user is entitled to for the specified registered query. A specific page of results can be requested along with a specific page size. Returns an array of the `bytes` representing the query results [structured content](#). Accepts three mandatory `string` arguments, two mandatory `int` arguments and one optional `Parameters` argument.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the <code>SessionService</code>).
NamedQuery	The name of the query required to be executed.
Page	The page of results which is required - first page is 1.
PageSize	The number of results to include on a single page of results.
Parameters	Allows query specific parameter values to be supplied to the query engine. This is an array of the Parameter type. Details are retrieved using the <code>NamedQueryService</code> GetParameters() method. Set this argument to <code>null</code> if there are no specific parameters.

Tally(ApplicationName, SessionId, NamedQuery, Page, PageSize, Parameters)

Retrieves a tally (group by / roll-up) of the results the currently logged in user is entitled to for the specified registered query. A specific page of results can be requested along with a specific page size. Returns an array of [Tally](#) objects. Accepts three mandatory `string` arguments, two mandatory `int` arguments and one optional `Parameters` argument.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the <code>SessionService</code>).
NamedQuery	The name of the query required to be executed.
Page	The page of results which is required - first page is 1.
PageSize	The number of results to include on a single page of results.
Parameters	Allows query specific parameter values to be supplied to the query engine. This is an array of the Parameter type. Details are retrieved using the <code>NamedQueryService</code> GetParameters() method. Set this argument to <code>null</code> if there are no specific parameters.

Tally (Object)

A data type used to return data group by / roll-up results provided by the query service. It provides details of the attributes being used in the tally and the actual result count.

Attributes	The list of the attributes used to generate the count. Each attribute in the list is a TallyAttribute .
Count	The count of unique items for the attribute values specified.

TallyAttribute (Object)

A data type used to return details of an attribute used to form a tally count.

AttributeName	The name of the attribute.
AttributeValue	The value of the attribute.
CategoryName	The category the attribute is defined for.

Parameter (Object)

See [details](#) shown in report service description.

Consuming the Query Service

To be able to execute registered queries and retrieve data, a client side proxy can be created to the `QueryService` provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the `QueryService`.

NOTE: Proxies *must* always be closed once their work is complete.

```
using (QueryServiceClient qsc = new QueryServiceClient(...))
{
    ...
}
```

IHS_Count Queries for Information Discovery

There are a number of pre-defined queries that are available to all users to facilitate information discovery across the entire IHS E&P Database, regardless of the data entitlements of the user. For example, these queries allow a user who subscribes only to the Middle East region to find out how many fields IHS has in Australia, or how many wells IHS has in an arbitrary lat / long bounding box. These queries are named with a prefix of "IHS_Count". Here are some examples:

```
IHS_CountContractsInBasin
IHS_CountContractsInCountry
IHS_CountContractsInLatLongBoundixBox
IHS_CountFieldsInBasin
```

IHS_CountFieldsInCountry
 IHS_CountFieldsInLatLongBoundixBox
 IHS_CountWellsInBasin
 IHS_CountWellsInCountry
 IHS_CountWellsInLatLongBoundixBox

This section shows how to use these IHS_Count queries to find out how many wells, fields and contracts IHS has for the “Gobustan-Absheron Trough (South Caspian Basin)” basin. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions.

```

...
using (QueryServiceClient qsc = new QueryServiceClient(...))
{
    List<Parameter> parameters = new List<Parameter>()
    {
        new Parameter()
        {
            Name = "BasinName",
            Type = "String",
            Values = "Gobustan-Absheron Trough (South Caspian Basin)"
        }
    };

    // see how many contracts there are matching the query
    int count = qsc.Count(
        applicationName,
        sessionId,
        "IHS_CountContractsInBasin",
        parameters);

    Console.WriteLine(
        "Total number of contracts available in basin = " + count);

    // see how many fields there are matching the query
    count = qsc.Count(
        applicationName,
        sessionId,
        "IHS_CountFieldsInBasin ",
        parameters);

    Console.WriteLine(
        "Total number of fields available in basin = " + count);

    // see how many wells there are matching the query
    count = qsc.Count(
        applicationName,
        sessionId,
        "IHS_CountWellsInBasin ",
        parameters);

    Console.WriteLine(
        "Total number of wells available in basin = " + count);
}
...

```

Running User Defined Queries

A single registered user query can be executed using three different methods which will produce different results.

To obtain a simple total count of available results, use the `Count()` method. This will return a single integer count of the number of data records which satisfy the criteria of the query.

To obtain the actual data results produced by executing the query, use the `Query()` method. This returns the results as an xml string conforming to the [Structured Content schema](#) represented as an array of `bytes`. The consumer is required to process the results into a meaningful representation for the consuming application.

To obtain a grouped count or data roll-up for the attributes listed as part of the query, use the `Tally()` method. This returns results as a list of `Tally` objects which contain detail of the unique attribute combinations available and the count of items within each of those combinations.

The code samples shown are based on the query shown below which has been registered using the name “IHS_TestQuery”.

```
<?xml version="1.0" encoding="UTF-8"?>
<Query xmlns="services.ihs.com/Search">
  <Attributes>
    <Attribute Entity="DBO_WELL_HEADER">COUNTRY_NAME_S</Attribute>
    <Attribute Entity="DBO_WELL_HEADER">WELL_CLASS_S</Attribute>
  </Attributes>
  <Source>
    <Categories Categories="DBO_WELL_HEADER"/>
  </Source>
  <Constraints>
    <Equals>
      <Attribute Entity="DBO_WELL_HEADER">COUNTRY_NAME_S</Attribute>
      <Value>
        <Literal>Cameroon</Literal>
      </Value>
    </Equals>
  </Constraints>
</Query>
```

This query is equivalent to the following SQL select statement:

```
SELECT COUNTRY_NAME, WELL_CLASS
FROM WELL_HEADER
WHERE (COUNTRY_NAME = 'Cameroon')
```

Below is an example of how to use the methods explained above and write the returned information to the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions.

```
...
using (QueryServiceClient qsc = new QueryServiceClient(...))
{
  // see how many records there are matching the query
```

```
int count = qsc.Count(
    applicationName,
    sessionId,
    "IHS_TestQuery",
    null);

Console.WriteLine(
    "Number of entitled results available = " + count);

// get a tally of the first 100 results
List<Tally> tallies = qsc.Tally(
    applicationName,
    sessionId,
    "IHS_TestQuery",
    1,
    100,
    null);

foreach (Tally t in tallies)
{
    foreach (TallyAttribute a in t.Attributes)
    {
        Console.WriteLine(
            "Attribute name = " + a.AttributeName +
            ", value = " + a.AttributeValue +
            ", category = " + a.CategoryName + ".");
    }
    Console.WriteLine("Tally count = " + t.Count);
    Console.WriteLine();
}

// run the query, get the first 100 results
int total;
byte[] results = qsc.Query(
    applicationName,
    sessionId,
    "IHS_TestQuery",
    1,
    100,
    null,
    out total);

// in the "real world" business processing would take place
// on these results to manipulate them into a form easily
// consumed by the human operator of an application, here
// the results xml is simply displayed on the console
using (StreamReader sr = new StreamReader(new MemoryStream(results)))
{
    Console.WriteLine("Results xml: " + sr.ReadToEnd());
}
...

```

Chapter 7 – Export

The **ExportService** provides supplies both details about the international spatial exports provided by IHS and the spatial exports themselves.

The **ExportService** provides access to a number of spatial layers such as wells, fields, contracts and blocks in spatial formats. This service provides methods to enable the consumer to discover which spatial layers are available, and what parameters are required to request each type of spatial layer.

To retrieve detailed information about the spatial exports available use the **GetExportTypes()** method which will retrieve details of all the available spatial exports. Detailed information is returned about the available export types including all available layers and details of all parameters.

To retrieve spatial export data the methods **Begin()**, **IsReady()**, **Export()** and **End()** must be used. The spatial export will be returned as an array of bytes in pages, the consumer must then reform this into a “document”.

NOTE: If a standard .NET auto-generated client and configuration is used then the binding default message size (64KB) and array length (16KB) will need to be modified as the default sizes will not be sufficient. The recommended setting to use is **5MB** for both properties. Depending on the internet connection speed available at the client the default send timeout may also need to be increased from 1 minute. These client settings may be increased / decreased as required in the future.

Figure 8 provides a visualization of what is available via the **ExportService**, Table 7 provides a detailed breakdown of the interface and the following sections provide C# code examples of using the service.

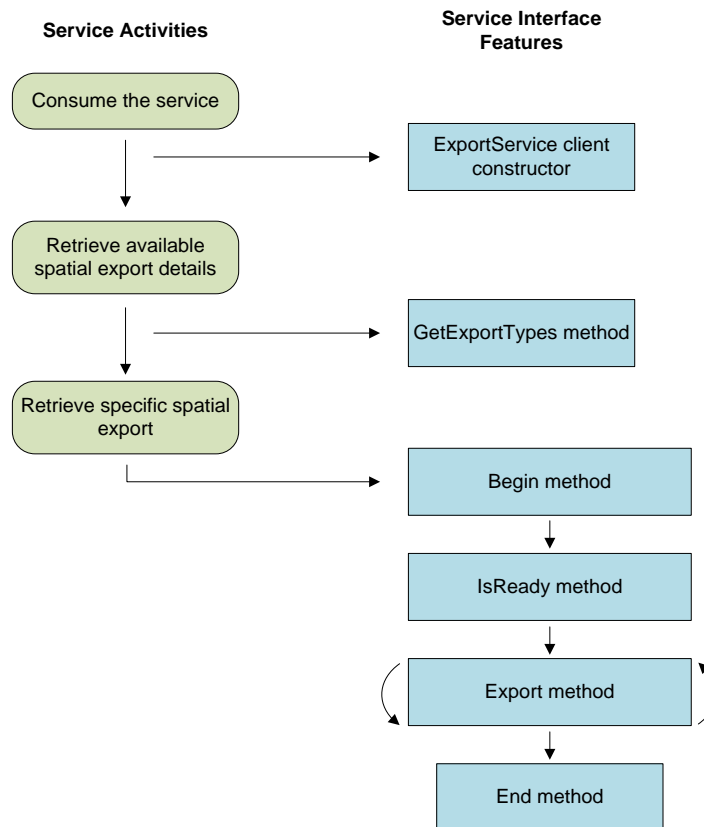


Figure 8 – The ExportService Interface

Table 7 — GraphService Interface Features

ExportService() + standard overloads

Creates a spatial export service object.

The name of the service used to access the spatial export services. A convention is to append “Client” to the name of the service to create a proxy instance of the service: **ExportServiceClient**.

GetExportTypes(ApplicationName, SessionId)

Retrieves details of available spatial exports. Returns an array of [ClientExportTypes](#). Accepts two mandatory arguments of type **string**.

ApplicationName Identifies the name of the application connecting to the Service.

SessionId The current session identifier (retrieved using the **SessionService**).

Begin(ApplicationName, SessionId, ExportFormat, NamedQuery, PageSize, Parameters, QueryType)

Starts the process of creating a spatial export file. Once **Begin(..)** has been called a matching **End(..)** *must* be supplied to prevent unnecessary persistence of export files.

This method returns a string export identifier to the caller which can be used in all subsequent calls to retrieve the spatial export.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
ExportFormat	Specifies the format key for the spatial export required. Retrieved from ClientExportType .Key via GetExportTypes .
NamedQuery	Specifies the query to use in order to extract a selection of data from the desired spatial export layer. E.g. in the case of using a BoundingBox to retrieve an export: a longitude, latitude pair supplied directly in the format (minimum longitude, maximum longitude)(minimum latitude, maximum latitude) – data is held to a precision of six decimal places.
PageSize	Specifies the page size (in bytes) to use when retrieving the spatial export.
Parameters	Adds restrictions to the spatial export being returned. E.g. the layer to retrieve the data from.
QueryType	Specifies the form in which the spatial export query is supplied and determines the format of the parameter NamedQuery . E.g. BoundingBox. NOTE: this is the only query type currently supported.

IsReady(ApplicationName, SessionId, ExportId)

Determines whether a spatial export is ready to be retrieved from the server.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
ExportId	The ExportId as returned from the Begin(...) operation.

Export(ApplicationName, SessionId, ExportId, PageNumber)

Retrieves the requested page of data for the specified spatial export.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the SessionService).
ExportId	The ExportId as returned from the Begin(...) operation.
PageNumber	The number of the export page to retrieve.

End(ApplicationName, SessionId, ExportId)

Completes the process of generating and downloading a spatial export. Informs the

server that the consumer has completed processing and that the export file being persisted on the server is no longer required. It is very important that all `Begin(..)` calls have a matching `End(..)` call.

ApplicationName	Identifies the name of the application connecting to the Service.
SessionId	The current session identifier (retrieved using the <code>SessionService</code>).
ExportId	The <code>ExportId</code> as returned from the <code>Begin(..)</code> operation.

ClientExportType (Object)

A data type used to return details about spatial export types supported by exports.

DataType	The data type of the export e.g. <code>PRODUCTION_DATA_TYPE</code> .
Description	A textual description of the spatial export type.
FileExtension	The file extension to be used for the export file which will be returned as an array of bytes as a result of calling <code>Export(..)</code> .
Key	The system key of the spatial export type.
Name	The displayable name of the spatial export type.
Parameters	A list of ExportTypeParameters describing the restriction parameters which can be applied to the spatial export.

ExportTypeParameter (Object)

A data type used to return details about data restriction parameters supported by spatial export types supplied by the export service.

keyField	The system key of the parameter.
suggestedValueField	Indicates the possible values for the parameter supported by IHS.
textField	The displayable name of the parameter.
Optional	Indicates whether the parameter is optional or mandatory.

Parameter (Object)

See [details](#) shown in report service description.

Consuming the Export Service

To access spatial export data, a client side proxy can be created to the `ExportService` provided by Enerdeq Web Services International. Below is an example of how to use a generated service proxy to create an instance to consume the `ExportService`.

NOTE: Proxies *must* always be closed once their work is complete.

```

using (ExportServiceClient esc = new ExportServiceClient(...))
{
    ...
}

```

Obtaining Details of Available Exports

To retrieve details about available spatial export types use the `GetExportTypes()` method. This method requires the name of the application being called and a valid session id. It returns an array of [ClientExportTypes](#).

[ClientExportType](#) describes all the details required to request a specific spatial export along with some more human readable information such as a description of the export type.

Below is an example of how the `GetExportTypes()` method can be used to retrieve spatial export details and display them on the console. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the return array type is represented as a generic list.

```

...
using (ExportServiceClient esc = new ExportServiceClient(...))
{
    // read export types
    List<ClientExportType> exports = esc.GetExportTypes(
        applicationName,
        sessionId);

    foreach (ClientExportType cet in exports)
    {
        Console.WriteLine(
            "Spatial Export name = " + cet.Name +
            ", key = " + cet.Key +
            ", description = " + cet.Description +
            ", file extension = " + cet.FileExtension +
            ", data type = " + cet.DataType + ".");

        foreach (ExportTypeParameter p in cet.Parameters)
        {
            Console.WriteLine(
                "Parameter key = " + p.keyField +
                ", name = " + p.textField +
                ", possible values = " + p.suggestedValueField + ".");
        }
    }
}
...

```

Retrieving a Spatial Export

To retrieve a spatial export use the `Begin()`, `IsReady()`, `Export()` and `End()` methods. The `Begin()` method requires details of the spatial export required and returns an export identifier to

be used to retrieve the export. The method takes eight mandatory arguments and one optional argument; these arguments specify the application name, a valid session id, spatial export format, data query, query type, desired page size and restriction parameters for the spatial export along with a possible specified file name to use for the export file (server side). The `IsReady()` method is used to determine whether the server has completed generating the export data for download. The `Export()` method is used to perform the download of spatial data from the server to the client. It returns an array of `bytes` which must be translated into a suitable format for human consumption by the service consumer e.g. file saved to hard disk. The extension to use for the saved file can be determined from the `ClientExportType` details for the export being performed.

Below is an example of how to use the export methods and write the returned spatial export to the hard disk. For brevity using the session service to log in and out although required has been omitted, please see the [session chapter](#) for details of how to perform these actions. The code below assumes that a `ClientExportType` object has been retrieved and selected as `selectedExport` previously. This code also demonstrates how a client can be configured to generate more easily consumable service calls; the parameter array type is represented as a generic list.

```
...
using (ExportServiceClient esc = new ExportServiceClient(...))
{
    // create the export layer restriction parameter
    List<Parameter> parameters = new List<Parameter>()
    {
        new Parameter()
        {
            Name = "SpatialLayerList",
            Values = "Wells"
        }
    };

    string exportId = null;

    try
    {
        // start the export process, supply bounding box coordinates
        // as the "named query"; page size is bytes
        BeginResponse beginResponse = esc.Begin(
            new esp.BeginMessage(
                applicationName,
                sessionId,
                selectedExport.Key,
                "(30,33) (30,33)",
                10240,
                parameters,
                "BoundingBox"));

        // poll for a ready initial export or a timeout
        exportId = beginResponse.ExportId;
        bool ready = false;
        int count = 0;
        const int Timeout = 10000;
        if (!ready && (count < Timeout))
        {
            count += 1000;
        }
    }
}
```

```

Thread.Sleep(1000);

IsReadyResponse readyResponse = esc.IsReady(
    new IsReadyMessage(applicationName, sessionId, exportId));

ready = readyResponse.IsReady;
}

// if ready get the export
if (ready)
{
    using (MemoryStream ms = new MemoryStream())
    {
        int page = 1;
        int totalPages;
        byte[] results;

        do
        {
            // grab the current page of results
            ExportResponse exportResponse = esc.Export(
                new ExportMessage(
                    applicationName,
                    sessionId,
                    exportId,
                    page));

            totalPages = exportResponse.TotalItemsAvailable;
            results = exportResponse.Results;

            if (results.Length > 0)
            {
                // dump the current export page into the memory stream
                ms.Write(results, 0, results.Length);
                page++;
            }
        } while (page <= totalPages);

        string appDataPath = Environment.GetFolderPath(
            Environment.SpecialFolder.ApplicationData);
        string fileName = "Export" + selectedExport.FileExtension;

        using (FileStream fs = new FileStream(
            Path.Combine(appDataPath, fileName),
            FileMode.OpenOrCreate))
        {
            using (BinaryWriter bw = new BinaryWriter(fs))
            {
                bw.Write(ms.ToArray());
            }
        }
    }
}
finally
{
    // *always* end the export process if it's been started
    if ((esc.State == CommunicationState.Opened) &&
        (!string.IsNullOrEmpty(exportId)))
    {
        esc.End(new EndMessage(applicationName, sessionId, exportId));
    }
}

```

```
}  
}  
...
```

Disclaimers

All code samples are provided “as is” and are provided to demonstrate using the service operation in as simple a way as possible. They include elements of hard coding which should not be used in a production system. The samples do not include exception handling, which should be included in a production system.

External links are provided throughout this document where relevant in order to provide helpful and useful information from the internet. IHS is not responsible for the content of external Internet sites.