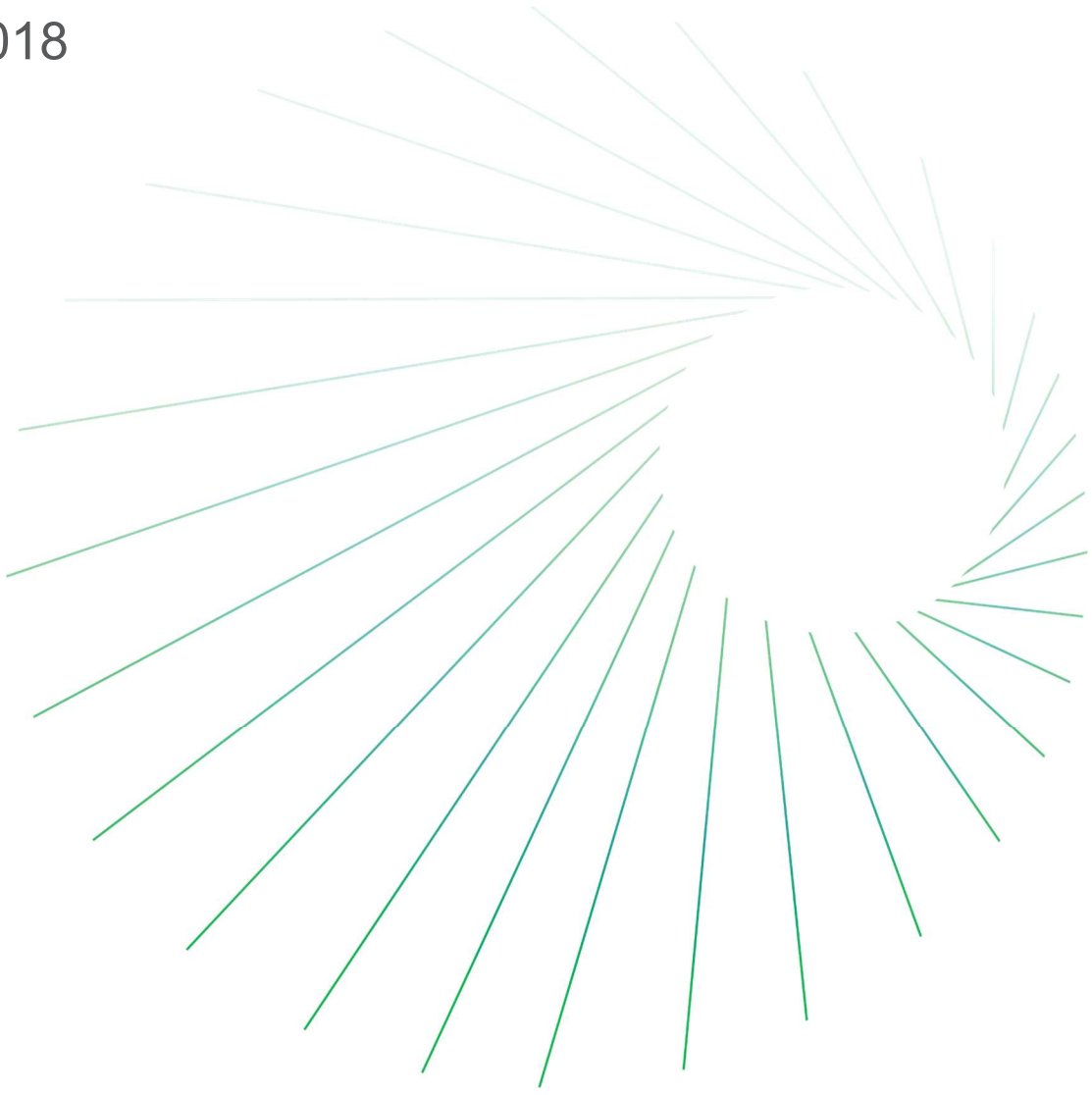
IHS Markit™

Energy Web Services North America 2018 v10

Developers Guide

13 December 2018



IHS Markit | **Developer Support Tool**

Energy Web Services North America 2018 v10 Developers Guide

December, 2018

Contents

Related Documents

IHS Markit Enerdeq Query Service CriteriaXML Specification provides details on defining queries used by the QueryBuilder service and is available for download from the Energy Web Services portal.

EnerdeqML schemas for Well and Production data are included with the SDK and are also available for download from the Energy Web Services portal.

US Data Export Formats for Well and Production data are available for download including samples from the Energy Web Services portal.

Chapter One

Overview

Energy Web Services North America enables access to IHS Markit US and Canadian Well, Production, Activity and Rig Activity content programmatically, through a message based API. These web services are accessed through supporting libraries available in both .NET 4.5.1 and Java 8. Alternatively, the provided **WSDL** documents can be used to directly generate proxies within your IDE or language of your choice.

These services can be accessed over the internet or with any Enerdeq intranet deployment. The base URL for the Internet (or hosted) version of the Energy Web Services is:

https://webservices.ihsenergy.com/WebServices.

The base URL for Intranet versions should be provided by the team deploying your individual version.

The set of available web services include:

Service	Description
Session	Handles authentication and validation of login credentials, along with company entitlements. https://webservices.ihsenergy.com/WebServices/Session
QueryBuilder	Search IHS Markit content for well, production, activity and rig activity data that meets the specified criteria. https://webservices.ihsenergy.com/WebServices/QueryBuilder
ExportBuilder	Builds structured data extracts for well and production, activity data and rig activity datatypes. https://webservices.ihsenergy.com/WebServices/ExportBuilder
GraphBuilder	Creates graph files of production data. https://webservices.ihsenergy.com/WebServices/GraphBuilder
ReportBuilder	Generates PDF reports for specified wells and production entities. https://webservices.ihsenergy.com/WebServices/ReportBuilder
DataTemplateManager	Manages custom data templates for use by the Export service. https://webservices.ihsenergy.com/WebServices/DataTemplateManager

A typical interaction between the client application, web services, and Enerdeq looks like this:

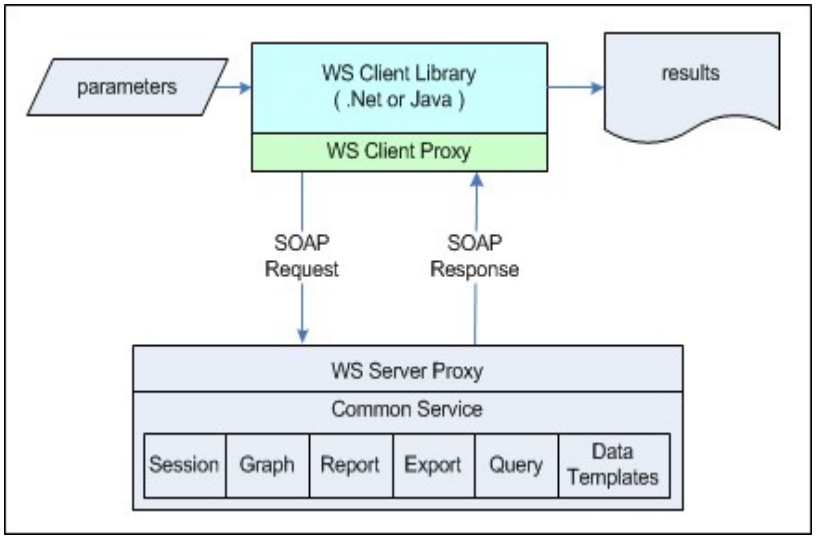


Figure 1 - Highlevel *Energy Web Services - US* architecture

The Export, Report, and Graph Services architecture is based on a request/acknowledge/poll and retrieve model. When the client application calls a web service to create a report, graph, or export, the web service returns a job ID. The client application is responsible for polling the status of the job ID until the job is completed. Upon job completion, the client application can retrieve the results of the web service request.

The Session, QueryBuilder and DataTemplateManager services implement the Request/Response client-service interaction pattern exclusively. The Export, Graph, and Report services implement the Request/Acknowledge/Poll interaction pattern for long running requests.

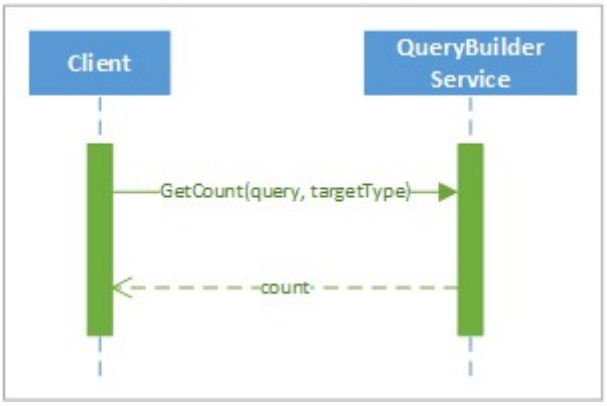


Figure 2: Request / Response Interaction

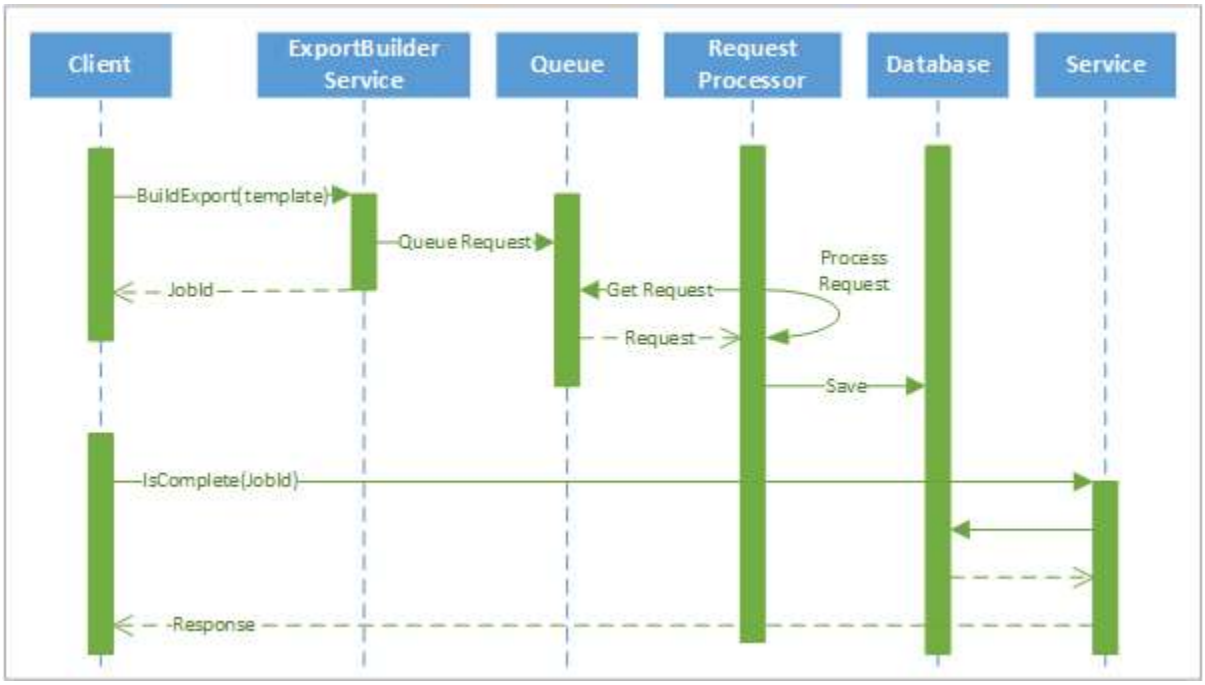


Figure 3: Request / Acknowledge / Poll Interaction

Chapter Two

Session

Session

The Session Web service handles authentication and authorization. By creating a session, authentication is localized in a single call. All subsequent API calls that occur during the session will not require further authentication, increasing the speed and efficiency of the Web Service conversation.

Starting with the 2017 v1 release, IHS Markit is white listing application keys. This means that you will need to contact IHS Markit Customer Care to register your application along with obtaining the necessary user credentials. Additionally, IHS Markit will provide application accounts for your Web Services solutions, these two can be requested through IHS Markit Customer Care.

To start a new session, use the **Create** method of the **WebServiceSession** class:

```
// override to hit staging deployment, by default session will be created
// against the Internet deployment.

string url = "https://webservices.ihsenergy.com/WebServices/Session";
string user = "<<USERNAME>>";
string pwd = "<<PASSWORD>>";
string app = "<<Your Application Key>>>";

if(url.Length > 0)
  ws = WebServicesSession.Create(url, user, pwd, _app);
else
  ws = WebServicesSession.Create(user, pwd, app);
```

When the session is complete, end the session with the **Destroy** method:

```
ws.Destroy();
```

The **WebServiceSession** class also provides the **getEntitlements** method for getting a company's entitlements. The method returns a string in XML format, which includes products information, and entitled regions of various different datatypes, including **PRODFit** and **Interpreted Tops**. Additionally, the **GetLatLongEntitlements** method is available to provide details on which Lat/Long sources a user subscribes to at a region level.

```
string entitlements = ws.GetEntitlements();
```

Chapter Three

Export Builder

ExportBuilder

The **ExportBuilder** service provides the primary way to access Well, Activity, Production and Rig Activity data in several formats suitable for loading into applications or databases. It offers structured data, one line, and spatial data exports.

Structured Data Exports

The **ExportBuilder** Service delivers structured data in a number of formats. The Export service uses a set of wells or production defined by a query or an ID list.

A **bridging** feature — bridging between the **Well** data and the **Production** data domains — makes it possible to request Production exports based on Well IDs (14-digit digit API Numbers) or Well exports based on Production IDs (IHS Markit Production IDs).

To create an **ExportBuilder** object, call the constructor for the **ExportBuilder** class, passing in a valid **WebServicesSession** object *ws*:

```
ExportBuilder eb = new ExportBuilder (ws);
```

The export format, which is specified in the template parameter in the Export Service, can be either a standard template or a customized, well-formed XML string. The standard templates only support the DELIMITER option (COMMA or FIXED FIELD). The standard Excel Work Book templates include all tabs. The standard EnerdeqML templates are xml formats without any filtering options set.

In 2018 v7 the **ExportBuilder** service will accept any well-formed XML string, **ExportXML**, passed to it with proper elements. Regardless of the spacing and indentation, if the XML string passed has all the proper elements, the data called for will be delivered.

Use the **GetTemplates** method to determine valid standard export templates for the given domain and datatype.

To request an export with the Build method using standard template:

```
string domain = "US";
string _datatype = "Well";
string _template ="EnerdeqML Well";
string[] _ids = {"13051231370001", "13051231370000};
string _file = "sample1";
Overwrite _ow = Overwrite.False;

string jobId = eb.Build(_domain, _datatype, _template, _ids, _file, _ow);
```

Use customized, well-formed XML strings, **ExportXML**, if more filtering options are needed. See **Appendix: ExportBuilder Data Filtering Options** for details on export template filtering options.

Note: If coding directly using the WSDL files rather than the Client Libraries (Java or .Net) then the XML template string has to be properly escaped, in particular the < and > symbols must be replaced by < and >.

This code snippet below builds exactly the same export as above, but it specifies the export format using customized XML template string, instead of a standard template name.

```
string domain = "US";
string datatype = "Well";
string template =     =
"<EXPORT><TEXTUAL_EXPORTS><WELL_XML></WELL_XML></TEXTUAL_EXPORTS></EXPORT>"
string[] ids = {"13051231370001", "13051231370000"};
string file = "sample2";
Overwrite ow = Overwrite.False;

string jobId = eb.Build(domain, datatype, template, ids, file, ow);
```

NOTE The domain specifies which Enerdeq database should be used to access data. Use the GetDomains method for a list of valid domains.

The datatype specifies what type of identifiers have been supplied in the ids parameter - well identifiers or production identifiers. Use the **GetDatatypes** method for a list of valid datatypes for the given domain.

The template parameter describes the format of the export. It can be either a standard template or a customized XML template string. See Appendix for details on customized XML templates. Use the **GetTemplates** method to determine valid standard export templates for the given domain and datatype.

The ids parameter is a list of IDs that describes which entities will be included in the export. For well data and activity data, use 14-digit digit API numbers, and for production unallocated data and production allocated data, use standard IHS Markit Production IDs. If a production export template is requested with well IDs, the Export Service uses the well IDs to identify the appropriate production entities to include in the export.

The filename parameter is used to name the export file built on the Enerdeq server, and the overwrite flag handles collision problems if the filename already exists on the server. If overwrite is set to False and the filename already exists, the export file will not be created. No file extension is allowed in the filename. The Export Service adds the file extension automatically based on the export format. See Appendix for details of file extensions.

To retrieve the list of available domains, data types, and standard templates, use these methods:

```
string[] domains = eb.GetDomains();
foreach (string domain in domains)
Console.WriteLine(domain);

string[] datatypes = eb.GetDatatypes(_domain);
foreach (string datatype in datatypes)
Console.WriteLine(datatype);

string[] templates = eb.GetTemplates(_domain, _datatype);
foreach (string template in templates)
Console.WriteLine(template);
```

To request an export with the **BuildFromQuery** method:

```
string domain = "US";
string datatype = "Well";
string template = "297 Well (Comma Delimited)";
string query = "My Query";
string file = "sample3";
Overwrite ow = Overwrite.False;

string jobId = eb.BuildFromQuery(domain, datatype, template, query, file, ow);
```

The **BuildFromQuery** method takes the name of a previously saved query or a **CriteriaXML** string. Details of **CriteriaXML** are described in the **IHS Markit Enerdeq CriteriaXML** document**. BuildFromQuery** will execute the query and use the resulting IDs to create the export. It bridges well IDs and production IDs if the data type of resulting IDs is well or activity data and the export format is production, and vice versa.

Standard Well Templates

297 Well (fixed format) US Domain only
297 Well (comma delimited) US Domain only
Excel Well Workbook (Excel 2002, 2003, 2007)
Excel Well Workbook (Excel 2007, 2010)
Excel Well Workbook (CSV)
Excel Directional Survey (Compatible with Excel 2003 or newer)
Excel Directional Survey (Compatible with Excel 2007 or newer)
Excel Directional Survey (CSV)

EnerdeqML Well
Well Header
Well Completion List
Well Completion List (CSV)
Well ID List
Geoscience Software PRODFit ASCII Export US Domain Only

Standard Production Templates

298 Production (comma delimited) US Domain only
298 Production (fixed field) US Domain only
298 Summary Production (comma delimited) US Domain only
298 Summary Production (fixed field) US Domain only
DMP2 Production US Domain only
DMP2 Summary Production US Domain only

EnerdeqML Production
Excel Production Workbook (Excel 2002, 2003, 2007)
Excel Production Workbook (Excel 2007, 2010)
Excel Production Workbook (CSV)
PowerTools Production Export US Doman only
PowerTools Production Export (comma delimited) US Domain only
PowerTools Summary Production Export US Domain only
Lease Producing Well Count (Compatible with Excel 2003 and newer) US Doman only
Lease Producing Well Count (Compatible with Excel 2007 and newer) US Doman only
Lease Producing Well Count (CSV) US Doman only
Production Header
Production ID List

Oneline Exports

The **ExportBuilder** Service supports Standard and Custom Oneline Exports via the **BuildOneline** and **BuildOnelineFromQuery** methods.

To request a Oneline export with the **BuildOneline** method follow the sample code below:

```
string domain = "US";
string datatype = "Well";
string[] ids = {"13051231370001", "13051231370000"};
string template = "Well Header List"; //A Standard or Custom Oneline template
name.
string file = "sample";
string filetype = "Excel";
Overwrite ow = Overwrite.False;

string jobId = eb.BuildOneline(domain, datatype, ids, template, filetype, file,
ow);
```

The *domain* specifies which Enerdeq database should be used to access data. Use the **GetDomains** method for a list of valid domains. This is especially important when requesting Canadian well and production data.

The *datatype* specifies what type of identifiers have been supplied in the *ids* parameter - well identifiers or production identifiers. Use the **GetDatatypes** method for a list of valid datatypes for the given domain.

The *ids* parameter is a list of IDs that describes which entities will be included in the export. For well data and activity data, use 14-digit digit API numbers, and for production unallocated data and production allocated data, use standard IHS Markit Production IDs. If a production export template is requested with well IDs, the Export Service uses the well IDs to identify the appropriate production entities to include in the export.

The *template* is the Oneline Standard or Custom Template name. Starting with version 1.11a the template can be the full XML definition. The definition of existing standard or custom templates can be retrieved using the **GetTemplateDefinition** method from the **DataTemplateManager**. Finally, a purely custom template definition can be created based on the schema provided in "**Appendix: Oneline Data Template Schema**". The lists of Well, Production, and Rig Activity Oneline Data Template attributes can be found on either the Enerdeq Browser site under Data Resource or one the Energy Web Services portal under Quick Reference.

The *filetype* is the output format of the oneline export. Use **GetFileTypes** method from the **DataTemplateManager** library for a list of valid file types.

The *file* parameter is used to name the export file built on the Enerdeq server. No file extension is allowed in the filename. The Export Service adds the file extension automatically based on the filetype (e.g. for Excel it is .xls, for HTML it is .html, for Tab Delimited it is .txt, and for Comma Delimited it is .csv).

The *overwrite* flag handles collision problems if the filename already exists on the server. If *overwrite* is set to False and the filename already exists, the export file will not be created.

Note: See the **Chapter 7 - DataTemplateManager** for details on data template management.

To request a Oneline export from a Query use the **BuildOnelineFromQuery** sample code below:

```
string query = "My Query";
string template = "Well ID List";
string file = "sample3";
string filetype = "Excel";
Overwrite ow = Overwrite.False;

string jobId = eb.BuildOnelineFromQuery(query, template, filetype, file, ow);
```

BuildOnelineFromQuery method takes the name of a previously saved query or a **CriteriaXML** string. Details of **CriteriaXML** are described in the **IHS Markit Enerdeq CriteriaXML** document**. BuildOnelineFromQuery** will execute the query and use the resulting IDs to create the export. It bridges well IDs and production IDs if the data type of resulting IDs is well or activity data and the export format is production, and vice versa.

Standard Well Data Templates

Well Header List
Well ID List
Operator/Lease List
Latitude/Longitude List
Activity Data Header List
Activity Data Production List

Standard Production Data Templates

Production Header List
Production ID List
Lease Volume List
Production IP/Normalized Data List
Operator Lease Volume List
Operator Volume List
Scoll List
Production Status List

Standard Activity Data Templates

Well Header List
Well ID List
Operator/Lease List
Latitude/Longitude List
Activity Data Header List
Activity Data Production List *only available for Activity Monitor users.

Standard Rig Activity Data Templates

Rig Activity Header List
Rig Activity Scroll List

Spatial Exports

The **ExportBuilder** Service also supports exports in shape file format, commonly known as a spatial export.

A spatial shape file export can be generated for a number of map layers within a specified lat/long bounding rectangle. Use the **GetLayers** method to retrieve a list of available spatial layers.

To request a spatial export with the **BuildSpatial** method:

```
double latMin = 31.283;
double latMax = 31.556;
double longMin = -92.985;
double longMax = - 92.575;
string[] layers = {"Congressional Section"};
string filename = "sample4";
bool overwrite = true;

string jobId = eb.BuildSpatial(latMin, latMax, longMin, longMax, layers,
                               filename, overwrite);
```

The *latMin* is minimum latitude.

The *latMax* is maximum latitude.

The *longMin* is minimum longitude.

The *longMax* is maximum longitude.

The *layers* are a list of supported map layers. Use the **GetLayers** method for a list of supported map layers. Only Valid layers (i.e., layers supported by **ExportBuilder** service, with correct syntax, entitled, and within the requested Lat/Long coordinates) will be returned in the generated Zip file.

There are some restrictions to the Lat/Long window:

1) Lat/Long Delta

   a) Lat Delta (i.e. difference between latMax and latMin) cannot be larger than 3.6. If it is larger, then the following error message is sent back:
   "Lat Max/Min delta is greater than the acceptable boundary of 3.6"

   b) Long Delta (i.e. difference between longMax and longMin) cannot be larger than 3.6. If it is larger, then the following error message is sent back:

   "Long Max/Min delta is greater than the acceptable boundary of 3.6"

Out of Bounds Lat/Long:

c) Latitude Bounds are:

i) latMin = -3.2. If submitted latMin is less than this value, the following error message is sent back:
"Submitted latMin value of *submittedLatMin* is outside the acceptable Lat bounds.
Acceptable Lat Min = -3.2 and acceptable Lat Max = 76.5"
NOTE: value in italic is the submitted value

ii) latMax = 76.5. If submitted latMax is greater than this value, the following error message is sent back:
"Submitted latMax value of *submittedLatMax* is outside the acceptable Lat bounds.
Acceptable Lat Min = -3.2 and acceptable Lat Max = 76.5"
NOTE: value in italic is the submitted value

d) Longitude Bounds are:

i) longMin = -167.8. If submitted longMin is less than this value, the following error message is sent back:
"Submitted longMin value of *submittedLongMin* is outside the acceptable Long bounds.
Acceptable Long Min = -167.8 and acceptable Lat Max = -66.4"

ii) NOTE: value in italic is the submitted value

iii) longMax = -66.4. If submitted longMax is greater than this value, the following error message is sent back:
"Submitted longMax value of *submittedLongMax* is outside the acceptable Long bounds.
Acceptable Long Min = -167.8 and acceptable Lat Max = -66.4"
NOTE: value in italic is the submitted value

If all requested Layers have incorrect Syntax or are not supported by ExportBuilder service, the following error message is sent back:

"There are No valid layers in the request"

If the submitted lat or long min/max values are reversed, i.e. if the submitted latMax is less than the submitted latMin, then the Energy Web Services - US assume that they have been accidently submitted this way and swap them.

The *filename* parameter is used to build the spatial export Zip file on the Enerdeq server. No file extension is allowed in the filename. The file could potentially be empty if the requested layers are not entitled or do not span the Lat/Long window submitted.

To retrieve the list of supported map layers, use **GetLayers** method:

```
string[] layers = eb.GetLayers();
foreach (string layer in layers)
  Console.WriteLine(layer);
```

The **Build**, **BuildFromQuery,** BuildOneline**, BuildOnelineFromQuery** and **BuildSpatial** methods return a jobID that can be used to track the progress of the export request until the job is complete:

```
while (eb.IsComplete(jobId == false))
{
      Console.Write(".");
      Thread.Sleep(2000);
}
```

Another way to track the Progress of the exports is via a **GetExportStatus** method that will return the current status of the export:

```
StatusType status = eb.GetExportStatus(jobId);

Console.WriteLine("Export file '{0}' has a status of '{1}'.", jobid, status)
```

To retrieve an export job with the **Retrieve** method:

```
bool compress = true;
byte[] byteArray = eb.Retrieve(jobId, compress);
```

If the Boolean *compress* is set to true, the Web Services compress the file before returning the compressed byte array. Set the compress to true when retrieving large export jobs. The client libraries always uncompress the byte array before returning the raw byte array.

 **Note:** For Spatial Exports, set the compress flag to false, as the export service automatically compresses and creates a zip file.

To delete the export job, use the **Delete** method:

```
eb.Delete(jobId);
```

To check whether an export job exists, use the **Exists** method:

```
eb.Exists(jobId);
```

To retrieve a list of completed export jobs, use the **List** method:

```
string[] _completed_exports = eb.List();

foreach (string completed_export in completed_exports)
      Console.WriteLine(completed_export);
```

Chapter Four

GraphBuilder

GraphBuilder

The GraphBuilder Service delivers image files of production history and pressure data graphs. Supported image formats include JPEG, PNG and PDF. A **bridging** feature — bridging between the **Well** data and the **Production** data domains — makes it possible to request Production graphs based on Well IDs (14-digit digit UWI Numbers).

To create a **GraphBuilder** object, call the constructor for the **GraphBuilder** class, passing in a valid **WebServicesSession** object *ws*:

```
GraphBuilder gb = new GraphBuilder (ws);
```

To request a graph with the **Build** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string template = "Rate vs. Time";
string[] idlist = {"235002670540"};
int height = 300;
int width = 300;
string format = "JPEG";
string file = "sample1";
bool ow = true;

string[] jobId = gb.Build(domain, datatype, template, idlist, height, width,
                          format, file, ow);
```

The *domain* specifies which Enerdeq database should be used to access data. Use the **GetDomains** method for a list of valid domains.

The *datatype* specifies what type of identifiers have been supplied in the *ids* parameter - well identifiers or production identifiers. Use the **GetDatatypes** method for a list of valid datatypes for the given domain.

The *template* parameter specifies the format of the graph data. Use the **GetTemplates** method to determine valid graph templates for the given domain and datatype. The **Build** method will build one image file per identifier provided in the *ids* parameter, unless the template selected is Summary Rate vs. Time, which will produce exactly one image file.

The *Height* and *Width* parameters define the pixel dimensions of the graph (IHS Markit recommends starting within the 300 – 500 pixel range for best results).

IDList is a list of production IDs or 14-digit digit API numbers that describes which entities will be included in the graph. If the data type of IDList is well or activity data, the Graph Service uses the well IDs to identify the appropriate production entities to include in the graph.

The *filename* parameter is used to build the graph file on the Enerdeq server, and the *overwrite* flag handles collision problems if the filename already exists on the server. If overwrite is set to False and the filename already exists, the

graph file will not be created. No file extension is allowed in the filename. The *Filename* parameter is a filename prefix in a real graph filename.

To retrieve the list of available domains, data types, image formats, and templates, use these methods:

```
string[] domains = gb.GetDomains();
foreach (string domain in domains)
  Console.WriteLine(domain);
Console.WriteLine();

string[] datatypes = gb.GetDatatypes(_domain);
foreach (string datatype in datatypes)
  Console.WriteLine(datatype);
Console.WriteLine();

string[] formats = gb.GetImageFormats();
foreach (string format in formats)
  Console.WriteLine(format);
Console.WriteLine();

string[] templates = gb.GetTemplates(_domain, _datatype);
foreach (string template in templates)
  Console.WriteLine(template);
Console.WriteLine();
```

To request a graph with the **BuildFromQuery** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string template = "Rate vs. Time";
string query = "My Query";
int height = 300;
int width = 300;
string format = "JPEG";
string file = "sample2";
bool ow = true;

string[] jobId = gb.BuildFromQuery(domain, datatype, template, query, height,
                                   width, format, file, ow);
```

The **BuildFromQuery** method takes the name of a previously saved query or a **CriteriaXML** string. Details of the **CriteriaXML** string are described in the **IHS Markit Enerdeq CriteriaXML** document. **BuildFromQuery** will execute the query and use the resulting IDs to create the graph. It bridges well IDs and production IDs if the data type of resulting IDs is well or activity data.

The **Build** and **BuildFromQuery** methods return an array of jobIDs that track the progress of each of the graphs built as the result of the request. Passing in a single ID results in a single jobID. Multiple IDs, and a summary graph results

in a single jobID. However, multiple IDs and a non-summary graph result in multiple jobIDs, which can be tracked for completion. To poll the status of a graph jobID:

```
while (gb.IsComplete(JobID[0]) == false)
{
    Console.Write(".");
    Thread.Sleep(2000);
}
```

Another way to track the progress of the graphs is via the **GetGraphStatus** method that will return the current status of the graph:

```
StatusType status = qb.GetGraphStatus(jobId);
Console.WriteLine("Graph file '{0}' has a status of '{1}'.", jobid, status)
```

To retrieve a graph job with the **Retrieve** method:

```
bool compress = true;
byte[] byteArray = gb.Retrieve(jobId[0], compress);
```

To delete the graph job, use the **Delete** method:

```
gb.Delete(jobId[0]);
```

To check whether a graph job exists, use the **Exists** method:

```
gb.Exists(jobId[0]);
```

To retrieve a list of completed graph jobs, use the **List** method:

```
string[] _completed_graphs = gb.List();

foreach (string completed_graph in completed_graphs)
Console.WriteLine(completed_graph);
Console.WriteLine();
```

Standard Production Graph Templates

Daily Rate vs. Time
Rate vs. Time
Pressure vs. Time
Pressure vs. Cumulative Gas
Summary Rate vs. Time

Chapter Five

ReportBuilder

ReportBuilder

The ReportBuilder Service delivers formatted report files of Well, Activity, Production and Rig Activity data. A bridging feature — bridging between the Well data and the Production data domains — makes it possible to request Production reports based on Well IDs (14-digit digit API Numbers) or Well reports based on Production IDs (standard IHS Markit Production IDs).

To create a **ReportBuilder** object in session, call the constructor for the **ReportBuilder** class, passing in a valid **WebServicesSession** object *ws*:

```
ReportBuilder rb = new ReportBuilder (ws);
```

To request a report with the **Build** method:

```
string domain = "US";
string datatype = "Well";
string template = "Well Summary Report (Portrait)";
string[] idlist = {"13051231370001", "13051231370000"};
string file = "sample";
bool ow = true;

string jobId = rb.Build(domain, datatype, template, ids, file, ow);
```

The *domain* specifies which Enerdeq database should be used to access data. Use the **GetDomains** method for a list of valid domains.

The *datatype* specifies whether the parameter ids are well data or production data. Use the **GetDatatypes** method for a list of valid datatypes for the given domain.

The *template* parameter specifies the format of the report. Use the **GetTemplates** method to determine valid report templates for the given domain and datatype.

IDList is a list of IDs that specifies which entities will be included in the report. For well and activity data, use 14-digit digit API numbers, and for production allocated and production unallocated data, use Production IDs. If a production export template is selected with well or activity data IDs, the **ReportBuilder** Service bridges well IDs and production IDs, and vice versa.

The *filename* parameter is used to build the report file on the Enerdeq server, and the *overwrite* flag handles collision problems if the filename already exists on the server. If the overwrite is set to **False** and the filename already exists, the report file will not be created. No file extension is allowed in the filename. The Report Service adds the file extension automatically.

To retrieve the list of available domains, data types, and templates, use these methods:

```
string[] domains = rb.GetDomains();
foreach (string datatype in datatypes)
      Console.WriteLine(datatype);

string[] datatypes = rb.GetDatatypes(_domain);
foreach (string datatype in datatypes)
      Console.WriteLine(datatype);

string[] templates = rb.GetTemplates(_domain, _datatype);
foreach (string template in templates)
      Console.WriteLine(template);
```

To request a report with the **BuildFromQuery** method:

```
string domain = "US";
string datatype = "Well";
string template = "Well Summary Report (Portrait)";
string query = "My Query";
string file = "sample";
bool ow = true;

string jobId = rb.BuildFromQuery(domain, datatype, template, query, file, ow);
```

The **BuildFromQuery** method takes the name of a query that is located on the Enerdeq server or a **CriteriaXML** string. Details of the **CrireriaXML** are described in the **IHS Markit Enerdeq CriteriaXML** document**. BuildFromQuery** will execute the query and use the resulting IDs to create the report. It bridges well IDs and production IDs if the data type of the resulting IDs is well or activity data and the export format is production, and vice versa.

The **Build** and **BuildFromQuery** methods return a jobID that tracks the progress of the report request. To poll the status of the jobID until the job is complete:

```
while (rb.IsComplete(JobID[0]) == false)
{
   Console.Write(".");
   Thread.Sleep(2000);
}
```

Another way to track the Progress of the graphs is via the **GetReportStatus** method that will return the current status of the graph:

```
StatusType status = rb.GetReportStatus(jobId);
```

```
Console.WriteLine("Report file '{0}' has a status of '{1}'.", jobid, status)
```

To retrieve the report job with the **Retrieve** method:

```
bool compress = true;
byte[] byteArray = rb.Retrieve(jobId, compress);
```

To delete the export job, use the **Delete** method:

```
rb.Delete(jobId);
```

To check whether a report job exists, use the **Exists** method:

```
rb.Exists(jobId);
```

To retrieve a list of completed report jobs, use the **List** method:

```
string[] completed_reports = rb.List();

foreach (string completed_report in completed_reports)
      Console.WriteLine(completed_report);
```

Well Report Templates

Scout Ticket (Portrait)
Scout Ticket (Landscape)
Well Summary Report (Portrait)
Well Summary Report (Landscape)
Production and Well Data Report (Portrait)

Production Report Templates

Abbreviated Report (Portrait)
Detailed Production Report (Portrait)
Detailed Well Test Report (Landscape)
Entity Well Report (Portrait)
Production/Injection Report (Portrait)
Production Report (Portrait)
Summary Production Report (Portrait)
Production and Well Data Report (Portrait)

Activity Data Report Templates

Scout Ticket (Portrait)
Scout Ticket (Landscape)
Well Summary Report (Portrait)
Well Summary Report (Landscape)

Rig Activity Report Templates

Rig Ticket (Portrait)
Rig Ticket (Landscape)
Rig Location (Landscape)
Rig Contractor (Landscape)
Rig Status Detail (Landscape)
Operator (Landscape)
Rig Location By Contractor (Landscape)

Chapter Six

QueryBuilder

QueryBuilder

The **QueryBuilder** service enables client applications to query the Web Services for items that match user-specified search criteria. A typical query would be "Find all entities that started producing oil in 2014 in the Anadarko Basin." Query criteria are specified in the form of **CriteriaXML**, which is detailed in the **IHS Markit Enerdeq Query Service - CriteriaXML Specfication** document. Queries that are saved using Enerdeq Browser can be used by the query service.

To create a **QueryBuilder** object in session, call the constructor for the **QueryBuilder** class, passing in a valid **WebServicesSession** object *ws*:

```
QueryBuilder qb = new QueryBuilder (ws);
```

A typical **CriteriaXML** query looks like this:

```
<criterias>
  <criteria type="value">
    <domain>US</domain>
    <datatype>Well</datatype>
    <attribute_group>Date</attribute_group>
    <attribute>Permit Date</attribute>
    <type>date</type>
    <displaytype></displaytype>
    <filter logic="less_than_or_equals">
      <value actual="1960/01/01"/>
    </filter>
  </criteria>
</criterias>
```

To retrieve a record count of the number of well, activity data, production allocated, or production unallocated entities that satisfy a query, pass the **CriteriaXML** query or the name of a previously saved query to the **GetCount** method:

```
int count = qb.GetCount(xmlQuery);
```

To retrieve a record count of the number of well, activity data, production allocated, or production unallocated entities that satisfy a query and a specified datatype, pass the **CriteriaXML** query or the name of a query that is already stored on the Enerdeq server and the specified datatype to the **GetCount** method:

```
string datatype = "production unallocated";
int count = qb.GetCount(xmlQuery, datatype);
```

If the specified datatype doesn't satisfy the query, the Enerdeq Query Service bridges well IDs and production IDs and returns a record count of the number of entities with the specified datatype.

Hint: We strongly suggest using the **BuildOnelineFromQuery** method in the **ExportService** to retrieve the equivalent content in a more performant manner instead of the **GetAttributes** method.

To retrieve header attributes for well or production entities that satisfy a query, use the **GetAttributes** method by specifying a **CriteriaXML** string or the name of a stored query on the Enerdeq server:

```
string attributes = qb.GetAttributes(xmlQuery);
```

The return value is a string XML representation of the attributes:

Well	Activity Data	Production	Rig Activity
Map Symbol	Map Symbol	Map Symbol	Map Symbol
Source	Source	Source	Contractor
Hole Direction	UWI	Hole Direction	Rig Name
UWI	API	Production ID	Rig Status
API	IC	Entity Type	Rig Sub-status
IC	Lease Name	Primary API	Current Depth
Regulator API	Well Num	Lease Name	UWI
Lease Name	Operator Name	Well Num	Lease Name
Well Num	Current Operator	Operator Name	Operator Name
Operator Name	Location	Location	Spud Date
Current Operator	Footage	Field Name	Location
Location	Field Name	State	Latitude
Footage	Region	County Name	Longitude
Field Name	State	OS Indicator	OS Area Code
Country Name	County Name	Basin	OS BLK
State	OS Indicator	Play Name	Field Name
County Name	Basin	Play Type	Rig Release Date
OS Indicator	Play Name	Status Name	Driller TD
Basin	Play Type	Resv Onshore	Form at TD Name
Play Name	Current Production Status	Resv Offshore	Proj Depth
Play Type	Final Status	Prod Zone Name	Proj Form
Current Production Status	Last Activity Date	Lease Code	Rig Type
Final Status	IP Prod Form Name	Oil Cum	Target Fluid
Final Status Code	Driller Td	Gas Cum	Latest Status Change
Last Activity Date	Log TD	Wtr Cum	Rig ID

Well	Activity Data	Production	Rig Activity
IP Prod Form Name	Form at TD Name	Oil YTD	Depth Capacity
Driller Td	Oldest Age Penetrated Name	Gas YTD	Draw Works
Log TD	Proj Depth	Wtr YTD	
Form At TD Name	Proj Form	Oil Latest Mo	
Oldest Age Penetrated Name	Target	Gas Latest Mo	
Oldest Age Penetrated Code	Permit License Date	Wtr Latest Mo	
Proj Depth	Permit Expiration Days	Max Production Volume – Gas	
Proj Form	Rig Onsite Date	Max Production Date - Gas	
Target	First Spud Date	Max Production Volume – Oil	
Permit License Date	Spud Date	Max Production Date - Oil	
First Spud Date	Comp Date	Max Production Volume – Water	
Spud Date	Suspended Date	Max Production Date - Water	
Comp Date	Abnd Date	Active Num Wells	
Initial Class	Initial Class	First Prod Date	
Lahee Class	Lahee Class	Last Prod Date	
Elevation Gr	Elevation Gr	Last Update Date	
Elevation Kb	Elevation Kb	TD	
Ref Elev	Ref Elev	TVD	
Elevation Code	Elevation Code	Upper Perf	
Latitude	Latitude	Lower Perf	
Longitude	Longitude	Oil Gatherer	
LL Srce	LL Srce	Gas Gatherer	
BH Latitude		Latitude	
BH Longitude		Longitude	
BH LL Srce		L and L Srce	
		BH Latitude	
		BH Longitude	
		BH LL Srce	

The XML schema of the return value is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="result-set">
```

```
    <xs:annotation>
      <xs:documentation>Comment describing your root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="record-meta">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="attribute" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="alias" type="xs:string" use="optional"/>
                  <xs:attribute name="name" type="xs:string" use="required"/>
                  <xs:attribute name="type" type="xs:string" use="required"/>
                  <xs:attribute name="function" type="xs:string"
use="optional"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="record" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="attribute"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="root" type="xs:string" use="required"/>
      <xs:attribute name="record-size" type="xs:long" use="optional"/>
      <xs:attribute name="record-count" type="xs:long" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

To retrieve a list of keys that match a query, use the **GetKeys** method, again by specifying a **CriteriaXML** string or the name of a stored query on the Enerdeq server:

```
string[] keys = qb.GetKeys(xmlQuery);
```

To retrieve a list of keys that match a query and a specified datatype, use the **GetKeys** method, again by specifying a **CriteriaXML** string or the name of a stored query and a datatype:

```
string datatype = "well";
string[] keys = qb.GetKeys(xmlQuery, datatype);
```

Working with Saved Queries

If the specified datatype doesn't satisfy the query, the Enerdeq Query Service bridges well IDs and production IDs, and returns a list of IDs with the specified datatype.

To obtain a list of queries on the Enerdeq server, use the **GetSavedQueries** method:

```
string[] queries = qb.GetSavedQueries();
```

To obtain a list of queries on the Enerdeq server for the given domain, use the **GetSavedQueries** method:

```
string domain = "US";
string[] queries = qb.GetSavedQueries(domain);
```

To obtain a list of queries on the Enerdeq server for the given domain and datatype, use the **GetSavedQueries** method:

```
string domain= "US";
string datatype = "production unallocated";

string[] queries = qb.GetSavedQueries(domain, datatype);
```

To obtain the **CriteriaXML** query definition of a previously saved query, use the **GetQueryDefinition** method:

```
string queryName = "queryName";
string criteriaXML = qb.GetQueryDefinition(queryName);
```

To save a query on the Enerdeq server, use the **SaveQuery** method:

```
string queryName = "queryName";
string criteriaXML = <the CriteriaXML query string>;
string result = qb.SaveQuery(queryName, criteriaXML);
```

To delete a query from the Enerdeq server, use the **DeleteQuery** method:

```
string queryName = "queryName";
string result = qb.DeleteQuery(queryName);
```

Lists of Valid Codes and Values

The **QueryBuilder** service also provides methods to populate lookup lists.

To get a list of valid lookup attributes per datatype use the **GetLookups** method. This method will return a list of lookup attributes along with the type of each.

If the client application needs to provide a list of valid names as a pick list, use the **LookupName** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string attr = "lease name";
string searchValue = "DOGwood";
SearchOperator operator = SearchOperator.StartsWith;

string[] names = qb.LookupName(domain, datatype, _attr, _searchValue,
_operator);
```

This example returns a list of lease names that start with "DOGwood".

The *domain* specifies which Enerdeq database should be used to access data.

The *datatype* specifies the subject area to look up.

The *attr* is a query attribute to look up, which is detailed in the **IHS Markit Enerdeq Query Service CriteraXML Specification** document**.** An empty name list is returned if the attribute has no name type.

The *searchValue* is search criteria. All of names are returned if search criteria are empty. An exception is thrown if the returned names exceed 750.

The *searchOperator* refines or expands the scope of the search.

If the client application needs to provide a list of valid name-code pairs as a pick list, use the **LookupNameCode** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string attr = "operator";
string searchValue = "Zebra Petroleum";
SearchOperator operator = SearchOperator.StartsWith;
NCType nctype = NCType.Name;

NameCode[] namecodes = qb.LookupNameCode(domain, datatype, attr, searchValue,
                                         operator, nctype);
```

This example returns a list of operator name-code pairs that start with "Zebra Petroleum".

The *domain* specifies which Enerdeq database should be used to access data.

The *datatype* specifies the subject area to look up.

The *attr* is a query attribute to be looked up, which is detailed in **the IHS Markit Enerdeq Query Service CriteraXML Specification** document**.** The attribute type should be either name or code.

The *searchValue* is search criteria. All of the names and codes are returned if search criteria are empty. An exception is thrown if return items exceed 2000.

The *SearchOperator* refines or expands the scope of the search.

The *nctype* indicates the attribute type of the search criteria. It can be either Name or Code.

If the client application needs to provide a list of state/county information as a pick list, use the **LookupStateCounty** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string searchValue = "AC NK";
SearchOperator operator = SearchOperator.StartsWith;
SCType sctype = SCType.CountyName;

StateCounty[] statecounties = qb.LookupStateCounty(domain, datatype,
searchValue, operator, sctype);
```

The *domain* specifies which Enerdeq database should be used to access data.

The *datatype* specifies the subject area to look up.

The *searchValue* is search criteria. All of state/county information is returned if search criteria are empty.

The *searchOperator* refines or expands the scope of the search.

The *sctype* indicates the type of the search criteria. It can be either CountyName or StateCode.

The returned state/county information includes state name, state code, county name and county code.

 **Note**: The lookup list methods can return only 2000 items. If the lookup produces more than 2000 items, an exception will be thrown.

If the client application needs to provide a list of query attribute code type information as a pick list, use the **LookupCode** method:

```
string domain = "US";
string datatype = "Production Unallocated";
string attr = "Operator";
string searchValue = "Zebra Petroleum";
SearchOperator operator = SearchOperator.StartsWith;

string[] codes = qb.LookupCode(domain, datatype, attr, searchValue, operator);
```

The *domain* specifies which Enerdeq database should be used to access data.

The *datatype* specifies the subject area to look up.

The *attr* is a query attribute to look up, which is detailed in the **IHS Markit Enerdeq Query Service CriteraXML Specification** document**.** An empty code list is returned if the attribute has no code type.

The *searchValu*e is search criteria. All of codes are returned if search criteria are empty. An exception is thrown if the number of codes found exceeds 2000.

The *searchOperator* refines or expands the scope of the search.

Miscellaneous Utility Methods

If a client application needs to provide a list of production entity codes and types for a list of uwis/entity codes and a date range, use the **GetEntityType** method:

```
string domain = "US";
string datatype = "production unallocated";
string startdate= "01/10/2006";string enddate= "01/15/2007";
string[] ids = {"14207C0105802", "14207C0131151", "14207C010580",
                "24207C174595"};

string result = qb.GetEntityType(_domain, datatype, ids, startdate, enddate);
```

The *domain* specifies which Enerdeq database should be used to access data.

The *datatype* specifies the parameter ids are well data or production data. The supported datatypes are Well, Activity Data, Production Allocated, and Production Unallocated.

The *ids* are a list of IDs that specifies which entities will be included in the **GetEntityType**. For well and activity data, use 14-digit digit API numbers, and for production allocated and production unallocated data, use Production IDs.

The *startdate* is the earliest last update date for the ids. The supported date format is yyyy/MM/dd. If it is set to empty or null, the default startdate is the earliest last updated date in the Enerdeq database.

The *enddate* is the latest last update date for the ids. The supported date format is yyyy/MM/dd. If it is set to empty or null, the default enddate is the latest last updated date in the Enerdeq database.

The returned *result* is an xml string containing production entity code and type information. See **Appendix C** for the **LookupXML** schema.

If a client application needs to provide a list of multiple attributes information such as code, name, and count for a list of well data or production data, use the **LookupAttributes** method:

```
string domain = "US";
string datatype = "production unallocated";
String[] attrs = {"formation", "leasename","pools/onshore","pools/offshore"};
string[] ids = {"142060005713", "242020049113","P18615"};

string result = qb.LookupAttributes(domain, datatype, attrs, ids);
```

The *datatype* specifies that the parameter ids are well data or production data. The supported datatypes are Well, Activity Data, Production Allocated and Production Unallocated.

The *attrs* are a list of query attributes to lookup. Currently the supported attribute for well and activity data is FORMATION. The supported attributes for production allocated and production unallocated data are FORMATION, LEASENAME (or LEASE NAME), POOLS/ONSHORE, and POOLS/OFFSHORE. Use other lookup methods to look up query attributes not supported in this method.

The *ids* are a list of IDs that specifies which entities will be included in the **LookupAttributes** results. For well and activity data, use 14-digit digit API numbers, and for production allocated and production unallocated data, use Production IDs.

The returned *result* is an xml string containing code, name, and count information. See Appendix: QueryBuilder LookupXML Schema for the LookupXML schema.

Detecting When Wells Have Been Deleted, or Had Their Identifier Changed

The **QueryBuilder** service exposes methods to help you detect and process certain key data management events:

"Change" is an event where a well identifier is changed. This includes IC to API number changes and situations where an API number is changed on a well.

"Delete" is an event where a well is permanently deleted. In some cases this is deletion of a well that has been erroneously stored twice. In other cases this is deletion of a well that was erroneously captured.

Here are statistics on number of events per day over 2016:

Average: 94 Median: 50 Max: 1651

There are two methods to identify and process these events:

GetChangesAndDeletes: Get details of all change and delete events within a date interval for the entire database. We recommend that you use this method if your target dataset has more than 1000 wells.

GetChangesAndDeletesFromIds: Get details of all change and delete events within a date interval for a specified list of well entities. We recommend that you use this method if your target dataset has less than 1000 wells.

Change/Delete Event Details

Here are the details returned for each change / delete event:

Attribute	Description
Uwi	Well Identifier for the well impacted by the change / delete event
Source	Source of data
Sequence	Record's sequence number
Date	Date UWI was changed or deleted
NewUwi	New UWI number to use
ReferenceUwi	Reference UWI refers to the pointer UWI. This is used in case of the removal of a well because it is duplicated under another UWI. This field identifies the well which exists in the database when the duplicate well is removed.
Remark	Description of change / delete event
ReasonCode	Code number associated with each unique Remark
ActiveCode	Value is Y or N. Y if the well has been either permitted, drilling in process, or completed. N if the well is classified as an Abandoned Location.
Proprietary	Value is Y or N. Y if there are proprietary data associated with the UWI.

Change / delete events must always be processed in the order specified by the Sequence attribute.

For certain change and delete event categories, the change / delete record includes both the "old" and "new" well identifier.

For the "CCC" events, the NewUwi value is populated.

For "DEL - WELL IS DUPLICATED UNDER ANOTHER API NUMBER" event, the ReferenceUwi value is populated.

The *NewUwi* and *ReferenceUwi* values will allow consumers to maintain referential integrity of foreign key references they have established to UWIs that are impacted by these change / delete events. This is particularly useful when other proprietary documents and information have been linked to an IHS Markit UWI that has been impacted by a change / delete event.

Note that for "DEL - NO SUCH WELL EXISTS" events, neither *NewUwi* nor *ReferenceUwi* is populated.

Change/Delete Event Reason Codes and Suggested Actions

A common use of *Energy Web Services - US* is to create and then maintain over time a well dataset that is synchronized nightly with the latest information from IHS. Here are the possible values for the Reason Code attribute in the change / delete record, and the suggested action you should take to keep your dataset synchronized.

Reason Code	Remark	Action
0	NOT CURRENTLY USED	None
1	CCC - WELL WAS ENTERED UNDER INCORRECT API NUMBER	Update your well record identified by *UWI* to be identified by *NewUwi.* Update any foreign key references to *UWI* to refer to *NewUwi.*
2	DEL - WELL IS DUPLICATED UNDER ANOTHER API NUMBER	Delete your well record identified by *UWI.* Update any foreign key references to *UWI* to refer to *ReferenceUwi.*
3	DEL – NO SUCH WELL EXISTS	Delete your well record identified by *UWI.*
4	DEL - WELL IS BEING READDED WITH NEW OR SAME API	Delete your well record identified by *UWI.*
5	CCC - API NUMBER CHANGE MADE BY REGULATORY AGENCY	Update your well record identified by *UWI* to be identified by *NewUwi.* Update any foreign key references to *UWI* to refer to *NewUwi.*
6	NOT CURRENTLY USED	None
7	CCC - CONTROL CODE CHANGE SHOULD NOT GO TO WHCS	Update your well record identified by *UWI* to be identified by *NewUwi.* Update any foreign key references to *UWI* to refer to *NewUwi.*
8	CCC - CHANGE IC ON WELL UNDER API CONTROL (ADD)	Update your well record identified by *UWI* to be identified by *NewUwi.* Update any foreign key references to *UWI* to refer to *NewUwi.*
9	CCC - CHANGE IC ON WELL UNDER IC CONTROL (ADD)	Update your well record identified by *UWI* to be identified by *NewUwi.* Update any foreign key references to *UWI* to refer to *NewUwi.*

Code Example

GetChangesAndDeletes method example:

```
string datatype = "Well";
string fromDate = "2010/06/01"
string toDate   = "2010/09/01"
int page = 1;
PagingResponse details;
string result = qb.GetChangesAndDeletes(datatype, fromDate, toDate, page, out
details);
```

The *datatype* specifies the type of Ids we're trying to find the changes and deletes on. Currently, the supported datatype is Well.

The *fromDate* is the earliest date for UWI changes. The supported date format is yyyy/MM/dd. If it is set to empty or null, the default fromDate is the earliest date supported by the *Energy Web Services - US.*

The *toDate* is the latest date for UWI changes. The supported date format is yyyy/MM/dd. If it is set to empty or null, then it will not be included in the criteria.

The *page* is the page number of the next set of data.

The *PagingResponse* is the object returned containing paging information about the results. It contains *Page*, *PageCount*, *Pages* and *TotalCount*.

Page is the current page number,

PageCount is the number of records in this Page,

Pages is the total number of Pages, and

TotalCount is the total number of records for all the pages.

GetChangesAndDeletesFromIds method example:

```
string datatype = "Well";
string[] ids = { "17085222020000","17715001550001","23163004310001" };
String fromDate = ""
String toDate   = ""
PagingResponse details;

string result = qb.GetChangesAndDeletesFromIds(datatype, ids, fromDate, toDate,
out details)
```

The *datatype* specifies the type of Ids we're trying to find the changes and deletes on. Currently, the supported datatype is Well.

The *fromDate* is the earliest date for UWI changes. The supported date format is yyyy/MM/dd. If it is set to empty or null, the default fromDate is the earliest date supported by Web Services.

The *toDate* is the latest date for UWI changes. The supported date format is yyyy/MM/dd. If it is set to empty or null, then it will not be included in the SQL query.

The *PagingResponse* is the object returned containing paging information about the results. It contains *Page*, *PageCount*, *Pages* and *TotalCount*.

Page is the current page number,

PageCount is the number of records in this Page,

Pages is the total number of Pages, and

TotalCount is the total number of records for all the pages.

Determine When Daily Update Is Complete

The database behind the services is updated daily. Typically, the update kicks off at 10pm MST and takes up to an hour to complete. The **QueryBuilder** service includes a method called **GetDailyUpdateInterval** which gives the date and time of the update. This method can be used when automating your daily incremental update.

Within the .NET client library, the method has been optimized as **GetLastUpdateInterval** returning an array of **DateTime** values indicating the start and end of the last update.

Chapter Seven

DataTemplateManager

The DataTemplateManager service provides functions to manage standard and custom oneline data templates.

To retrieve the list of available filetypes, standard templates, custom templates, and all templates, you will need to use these methods available via the DataTemplateManager:

```
DataTemplateManager dtm = new DataTemplateManager(ws);

string[] fileTypes = dtm.GetFileTypes();
foreach (string fileType in fileTypes)
Console.WriteLine(fileType);
Console.WriteLine();

string[] templates = eb.GetStandardTemplates(domain, datatype);
foreach (string template in templates)
Console.WriteLine(template);
Console.WriteLine();

string[] templates = eb.GetCustomTemplates(domain, datatype);
foreach (string template in templates)
Console.WriteLine(template);
Console.WriteLine();

string[] templates = eb.GetTemplates(domain, datatype);
foreach (string template in templates)
Console.WriteLine(template);
Console.WriteLine();
```

To retrieve any data template definition, save or delete a custom oneline data template, you will need to use these methods available via the DataTemplateManager:

```
String template = "Well ID List"; //The Oneline Template Name.
DataTemplateManager dtm = new DataTemplateManager(ws);

//// RETRIEVING A ONELINE TEMPLATE DEFINITION
Console.Writeline(”Getting the Oneline Template Definition: ” + template);
String definition = dtm.GetTemplateDefinition(template);


//// SAVING A CUSTOM ONELINE TEMPLATE
//*************************************************************
// Read in the Custom Template
//*************************************************************
try
{
customTemplate = File.ReadAllText(template);
}
catch (Exception e)
{
Console.WriteLine("Can't read file {0}: {1}", template, e.Message);
return;
}
//*************************************
// Save a Custom Template.
//*************************************
dtm.SaveTemplate(customTemplate);
Console.WriteLine("Custom Template file " + customTemplate + " is saved”);


//// DELETING A CUSTOM ONELINE TEMPLATE
//*************************************
// Delete a Custom Template.
//*************************************
Console.WriteLine("Deleting Custom Template : " + template);
dtm.DeleteTemplate(template);
Console.WriteLine("Custom Template file " + template + " is deleted");
```

Chapter Eight

Programming Guidelines and Useful Resources

Programming Guidelines

Passing the Session Cookie from one Web Service to Another

There is c# sample (Program.cs) which provides you with sample code of how to how to pass the Session cookie from one Web Service to another. It is available in the .NET client libraries under Samples.

Submit Large Export and Report Requests in "Chunks"

Internal and external experience makes us strongly recommend that all large export requests are broken down into a number of smaller "chunks", and that the "chunk size" be a configurable parameter. For example, if you have a list of 10,000 wells for which you want to generate a 297 export, then break that request in to 10 requests each of 1,000 wells.

Within the services requests are prioritized by size, requests for 1-1999 ids in size receive the highest priority, next 2000-59999 ids receive lower priority, and requests over 60000 ids in size receive the lowest priority.

Always Understand the Size of Requests Before You Make Them

Always use getCount() before using getAttributes(), getKeys(), buildExportFromQuery(), etc.

Don't Use getAttributes() or getKeys() for More Than 5000 Records

If getCount() returns more than 5000, then either refine the query until it returns less than 2000, or use buildExportFromQuery() instead of getAttributes() or getKeys(). buildExportFromQuery() performs much faster than getAttributes() or getKeys(). To emulate getKeys() with buildExportFromQuery(), use the “Well ID List” export template. To emulate getAttributes() with buildExportFromQuery(), use the “Well Header” export template.

The Application Parameter

There is a parameter "application" in a number of the methods. Please always use the value provided to you by IHS Markit so that IHS Markit operations can identify which application or software is calling the Web Services.

Always Use the WebServicesSession Service

Although the API technically allows interaction with DataTemplateManager, ReportBuilder, ExportBuilder, GraphBuilder and QueryBuilder services without using the WebServicesSession service, this is only provided for backwards-compatibility purposes. Please create a session as the first step to your interaction.

Use Appropriate Sleep Times between Polls

Always sleep for at least 2 seconds between each poll for status of build export, report and graph outputs. Use good judgment and sleep longer for larger requests, especially for longer running, non-user-interactive data management processes.

Always Compress Large Export Jobs on Retrieval

Always set the compress flag to true when retrieving large exports. This will make retrieval much faster. Note that the Excel exports are already in a zip file so do not need to be compressed on retrieval.

How to Decompress the Output from Retrieve () when using WSDL.

Starting with .NET 4.5.1 Microsoft offers decompression of zip files natively using the ZipArchive class in the System.IO.Compression namespace. The same holds true for Java using the java.uitl.zip package.

Delete Jobs after Retrieval

After you've successfully retrieved a file, send a 'Delete' request to remove the job from our servers. In early 2015 IHS Markit introduced a 1GB limit on the home volume of each account without deleting successful retrieval you will quickly fill up this space and further requests will fail.

```
//create a ExportBuilder object
eb = new ExportBuilder(session);

//delete export file
boolean done = eb.delete(_jobID);
```

Multithreaded Code

In multithreaded applications the ExportBuilder, GraphBuilder or ReportBuilder should be created within each thread and not shared between threads. There is an issue specific to retrieving files with the compress flag set to true.

Useful Resources

Web Services Portal

The Web Services Portal (https://webservicesportal.ihsenergy.com) details both IHS Markit and 3rd party applications that use these Web Services. In addition, documentation, samples and schemas are available to support the development experience.

Reference Material

Documentation of export formats, including EnerdeqML, can be found on the Data Export tab here:

https://ihsmarkit.com/products/oil-gas-reference-materials.html#tab-6

Documentation of lists of valid values, such as well statuses, can be found at the same page on the Reference Code Abbreviations tab.

Typical Workflow

A typical workflow supported by the Energy Web Services is that of creating and maintaining a project. To illustrate how the services can be used together we will start by defining our project as all well and production data contained within a lat/long extent.

The first step is the initial download, followed by incremental updates.

Initial Download

To begin, instantiate a WebServicesSession:

```
string url = "";
string user = "<<Your username>>";
string pwd = "<<Your password>>";
string app = "<<<Your Application Name>>>";

WebServicesSession ws;
if(_url.Length > 0)
```

```
ws = WebServicesSession.Create(_url, _user, _pwd, _app);
else
ws = WebServicesSession.Create(_user, _pwd, _app);
```

The project definition or scope needs to be expressed in a CriteriaXML query. In the example below, the CriteriaXML requests all well IDs in a lat/long bounding rectangle:

```
string xml = "";
xml += "<criterias>";
xml += "  <criteria type=\"group\">";
xml += "    <domain>US</domain>";
xml += "    <datatype>Well</datatype> ";
xml += "    <attribute_group>Grid/Location</attribute_group>";
xml += "    <attribute>Lat/Long-Surface</attribute> ";
xml += "    <filter logic=\"include\">";
xml += "      <value>";
xml += "        <group_actual>";
xml += "          <operator logic=\"and\">";
xml += "            <condition logic=\"between\">";
xml += "              <attribute>latitude</attribute> ";
xml += "              <value_list>";
xml += "                <value min=\"42.483\" max=\"42.5125\" /> ";
xml += "              </value_list>";
xml += "            </condition>";
xml += "            <condition logic=\"between\">";
xml += "              <attribute>longitude</attribute> ";
xml += "              <value_list>";
xml += "                <value min=\"-109.8\" max=\"-109.6\" /> ";
xml += "              </value_list>";
xml += "            </condition>";
xml += "          </operator>";
xml += "        </group_actual>";
xml += "      </value>";
xml += "    </filter>";
xml += "  </criteria>";
xml += "</criterias>";
```

To validate the CriteriaXML and at the same time get a count of IDs use the the QueryBuilder getCount() method as shown below:

```
Console.WriteLine("Creating QueryBuilder Object...");
QueryBuilder qb = new QueryBuilder (ws);
Console.WriteLine("QueryBuilder Object Created. ");

Console.WriteLine("Executing Query...");
int count = qb.GetCount(xml);
```

```
Console.WriteLine("Query returned " + count.ToString() + " well records...");
```

The “EnerdeqML Well” format is one good candidate for downloading IHS Markit well data to a project database. The following code snippet demonstrates the export procedure:

```
Console.WriteLine("Creating ExportBuilder Object...");
ExportBuilder eb = new ExportBuilder (ws);
Console.WriteLine("ExportBuilder Object Created.");
Console.WriteLine();

Console.Write("Requesting Well Export...");
string ExportJobID = eb.BuildFromQuery("US", "Well", "EnerdeqML Well", xml,
"sample_1", Overwrite.True);

while (eb.IsComplete(ExportJobID) == false)
{
      Console.Write(".");
        Thread.Sleep(2000);
}
Console.WriteLine("Well Export Created.");

Console.WriteLine("Retrieving Well Export...");
byte[] ExportData = eb.Retrieve(ExportJobID, true);
Console.WriteLine("Well Export Retrieved: " + ExportData.Length.ToString() +
                          " bytes");
```

A **bridging** feature — bridging between the **Well** data and the **Production** data domains — makes it possible to request Production graphs, exports, and reports based on Well IDs. The example below shows how to generate production export and report for all well IDs (API Number) in a lat/long bounding:

To get a count of production IDs from a well query instantiating the QueryBuilder, call the getCount() method as shown below:

```
Console.WriteLine("Start to bridge Well IDs and Production IDs...");

Console.WriteLine("Retrieving a production record count by executing a well
query...");
count = qb.GetCount(xml, "Production Unallocated");
Console.WriteLine("Query returned " + count.ToString() + " production
records...");
```

To request a “EnerdeqML Production” export with a well query, use the following methods call:

```
Console.Write("Requesting Production Export using a production template and the
well query...");
ExportJobID = eb.BuildFromQuery("US", "well", "298 Production (Comma
Delimited)", xml, "sample_3", Overwrite.true);

while (eb.IsComplete(ExportJobID) == false)
{
Console.Write(".");
Thread.Sleep(2000);
}
Console.WriteLine();
Console.WriteLine("Production Export Created.");
Console.WriteLine();

Console.WriteLine("Retrieving Production Export...");

ExportData = eb.Retrieve(ExportJobID, true);
Console.WriteLine("Production Export Retrieved: " +
ExportData.Length.ToString() + " bytes");
Console.WriteLine();
```

Well Incremental Updates

The IHS Markit E&P database gets updated every night and we need to be able to keep the project refreshed with the latest data on a periodic basis.

There are two steps required to synchronize. Firstly, you need to detect and process UWI change and delete events. Then you need to detect and process new well records and well records with updated information.

Detect and Process Well Identifier Change and Delete Events

Define a change / delete query that returns the UWIs that have been changed or deleted since last refresh. The result contains the details of each change / delete event, and the sequence the events were processed into our database. It is important that you process the records in the same sequence.

Get the UWIs that have been changed or deleted.

```
Console.WriteLine("Creating QueryBuilder Object...");
qb = new QueryBuilder(ws);
Console.WriteLine("QueryBuilder Object Created.\n");

// Retrieve Change/Delete records

Console.WriteLine("Retrieving change/delete records ...");

string yesterday = (DateTime.Today.AddDays(-1)).ToString("yyyy/MM/dd");
Console.WriteLine("yesterday = " + yesterday);

string today = DateTime.Today.ToString("yyyy/MM/dd");
Console.WriteLine("today = " + today);

PagingResponse details;
int page = 1;
int pages;
do
{
string result = qb.GetChangesAndDeletes("Well", yesterday,
                                        today, page, out details);

// Loop through each page of results

page = details.Page;
pages = details.Pages;
Console.WriteLine("--- Page {0} ---", page);
Console.WriteLine("Page:{0} PageCount:{1} Pages:{2}
                TotalCount:{3}\n\n", details.Page, details.PageCount,
                details.Pages, details.TotalCount);

Console.WriteLine("\n{0}", FormatXml(result));
page++;
} while (page <= pages);
```

Detect and Process New Well Records and Well Records with Updated Information

We can define a query that identifies wells added or updated in the project lat/long area of interest with a Last Update criteria added to constrain the result set to only include records modified or added to the underlying database since last refresh.

The CriteriaXML could be specified as follow:

```
xml  = "<criterias>";
xml += "  <criteria type=\"group\">";
xml += "    <domain>US</domain>";
xml += "    <datatype>Well</datatype> ";
xml += "    <attribute_group>Grid/Location</attribute_group>";
xml += "    <attribute>Lat/Long-Surface</attribute> ";
xml += "    <filter logic=\"include\">";
xml += "      <value>";
xml += "        <group_actual>";
xml += "          <operator logic=\"and\">";
xml += "            <condition logic=\"between\">";
xml += "              <attribute>latitude</attribute> ";
xml += "              <value_list>";
xml += "                <value min=\"42.483\" max=\"42.5125\" /> ";
xml += "              </value_list>";
xml += "            </condition>";
xml += "            <condition logic=\"between\">";
xml += "              <attribute>longitude</attribute> ";
xml += "              <value_list>";
xml += "                <value min=\"-109.8\" max=\"-109.6\" /> ";
xml += "              </value_list>";
xml += "            </condition>";
xml += "          </operator>";
xml += "        </group_actual>";
xml += "      </value>";
xml += "    </filter>";
xml += "  </criteria>";
xml += "  <criteria type=\"value\">";
xml += "    <domain>US</domain>";
xml += "    <datatype>Well</datatype>";
xml += "    <attribute_group>Date</attribute_group>";
xml += "    <attribute>Last Update</attribute>";
xml += "    <type>date</type>";
xml += "    <filter logic=\"greater_than_or_equals\">";
xml += "      <value actual=\"$$DATE$$\" />";
xml += "    </filter>";
xml += "  </criteria>";
xml += "</criterias>";
```

Production Incremental Updates

A production entity could be added or deleted. The production entity ID could be amended or the content be updated. In any case, a production entity ID should always tie to a well API at all time. Therefore production incremental updates require well incremental update process above.

To synchronize well and production data with IHS Markit data, here is the recommended workflow:

1. Detect and Process Well Identifier Change and Delete Events
2. Detect and Process New Well Records and Well Records with Updated Information
    a. If a new well is added in IHS Markit database (meaning it exists in IHS Markit database but doesn't exist in local database), add the new well and export all its production data and add to local database.
    b. If a well is updated this means either the well content is changed or the production entity ID associated with that well is changed. The synchronization process should handle both cases.

To synchronize the production data:

1. Identify and remove obsolete production IDs that are in the local database but do not exist in IHS Markit database.
2. Detect and process new production records or production records with updated content.

Identify and remove obsolete production IDs

This can be done by well API or in batch. To synchronize the production ID, get a list of production ID (a well or all production ID for batch process), then remove production ID that are in local database but doesn't exist in IHS

Detect and Process New Production Records and Production Records with Updated Information

We can define a query that extracts production IDs added or have content updated with the Last Update criteria. If the production ID is newly added, extract its entire production history. If the production ID is amended, the start/end date range could be used to streamline the producing year/month. This can be done by well or in batch.

Chapter Eight

Appendix A: ExportBuilder Data Filtering Options

TEXTUAL EXPORTS

Supported textual exports are:

Value	Description	Extension
WELL_297	297 export for Well data	.97c or .97f
WELL_XML	EnerdeqML Well export	.xml
PRODUCTION_298	298 export for Production data	.98c or .98f
PRODUCTION_DMP2	DMP2 export for Production data	.DP2
PRODUCTION_XML	EnerdeqML Production export	.xml
WELL_WORKBOOK	Well Excel workbook	.zip
PRODUCTION_WORKBOOK	Production Excel workbook	.zip

The attributes that can be assigned to each type of export are:

WELL_297 Attributes	
DELIMITER	Possible values COMMA, FIXED_FIELD. Default = COMMA
EX_MISSING_LOCATIONS	This attribute is used to exclude wells from the export that are missing lat/long locations. Possible values are TRUE or FALSE. Default is FALSE.
RECORDS	To be used to export only some of the records. Default = 0, all records. The value of each record is expressed in HEX format. Convert the values to decimal form and add up the values of the records that you want included.

Record Name	Value
ALL_RECORDS	0x00000000
HEADER	0x00000001
FORMATION	0x00000002
IP	0x00000004
IP_DETAILS	0x00000008
PT	0x00000010
PT_DETAILS	0x00000020
DST	0x00000040
CORE	0x00000080
LOG	0x00000100
MUD	0x00000200
CSG_TBG_LNR	0x00000400
LOC_NARR	0x00000800
DRILL_NARR	0x00001000
PROP_BHL	0x00002000
ACTUAL_BHL	0x00004000
PROP_REF_BHL	0x00008000
ACTUAL_REF_BHL	0x00010000
DEV_SRVY	0x00020000
DIR_SRVY	0x00040000
HRZ_GEN	0x00080000

	HRZ_SRVY	0x00100000
	HRZ_KOP	0x00200000
	HRZ_POE	0x00400000
	HRZ_SPOKE	0x00800000
	START_END_ALL	0x01000000

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
            <WELL_297 DELIMITER='COMMA'/>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

WELL_XML Attributes	
VERSION	Version of XML Schema to export. This option is reserved for future use. Currently, the latest version is always exported, regardless of what is specified here.
EXCLUDE_MISSING_LATLONGS	This attribute is used to exclude wells from the export that are missing lat/long locations. Possible values are TRUE or FALSE. Default is FALSE.
INCLUDE_PRODFIT	This attribute is used to include PRODFit data in the EnerdeqML where PRODFit data is available. Possible values are TRUE or FALSE.
INCLUDE_SUBSCRIBED_LATLONG_SOURCES	This attribute is used to include all subscribed lat/long sources for a well. Possible values are TRUE, or FALSE.
<BRANCH NAME=…>	The following branches can be specified to be included:
	/WELL_SET/WELLBORE/METADATA
	/WELL_SET/WELLBORE/HEADER
	/WELL_SET/WELLBORE/LOCATION
	/WELL_SET/WELLBORE/SURVEYS
	/WELL_SET/WELLBORE/MECHANICAL
	/WELL_SET/WELLBORE/GEOLOGY
	/WELL_SET/WELLBORE/GEOPHYSICS
	/WELL_SET/WELLBORE/ENGINEERING

	/WELL_SET/WELLBORE/TESTS /WELL_SET/WELLBORE/CONTENT

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
                  <WELL_XML EXCLUDE_MISSING_LATLONGS='TRUE'
                  INCLUDE_PRODFIT='TRUE'
                  INCLUDE_SUBSCRIBED_LATLONG_SOURCES='TRUE'>
                  <BRANCH NAME='/WELL_SET/WELLBORE/HEADER'/>
                  <BRANCH NAME='/WELL_SET/WELLBORE/LOCATION'/>
            </WELL_XML>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

PRODUCTION_298 Attributes	
DELIMITER	Possible values COMMA, FIXED_FIELD. Default = COMMA
START_MONTH	Number of month to start reporting production. Default = 0. In 2 digits format. START_MONTH and START_YEAR should be presented together.
START_YEAR	Number of year to start reporting production. Default = 0. In 4 digits format. START_MONTH and START_YEAR should be presented together.
END_MONTH	Number of month to stop reporting production. Default = 0. In 2 digits format. END_MONTH and END_YEAR should be presented together.
END_YEAR	Number of year to stop reporting production. Default = 0. In 4 digits format. END_MONTH and END_YEAR should be presented together.
TESTS	If = 'T' export test records, if = 'F' tests records not exported. Default = 'T'.
COMBINED	If = 'T' summarize all entities, if = 'F' export each entity . Default = 'F'.

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
            <PRODUCTION_298 DELIMITER='COMMA' TESTS='F' COMBINED='T'/>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

PRODUCTION_DMP2 Attributes	
START_MONTH	Number of month to start reporting production. Default = 0. In 2 digits format. START_MONTH and START_YEAR should be presented together.
START_YEAR	Number of year to start reporting production. Default = 0. In 4 digits format. START_MONTH and START_YEAR should be presented together.
END_MONTH	Number of month to stop reporting production. Default = 0. In 2 digits format. END_MONTH and END_YEAR should be presented together.
END_YEAR	Number of year to stop reporting production. Default = 0. In 4 digits format. END_MONTH and END_YEAR should be presented together.
TESTS	If = 'T' export test records, if = 'F' tests records not exported. Default = 'T'.
COMBINED	If = 'T' summarize all entities, if = 'F' export each entity . Default = 'F'.

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
            <PRODUCTION_DMP2/>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

PRODUCTION_XML Attributes	
VERSION	Version of XML Schema to export. This options is reserved for future use. Currently, the latest version will always be exported regardless of what is specified here.
COMBINED	If = 'T' summarize all entities, if = 'F' export each entity . Default = F
START_MONTH	Number of month to start reporting production. Default = 0, in 2 digits format. START_MONTH and START_YEAR should be presented together.
START_YEAR	Number of year to start reporting production. Default = 0, in 4 digits format. START_MONTH and START_YEAR should be presented together.
END_MONTH	Number of month to stop reporting production. Default = 0, in 2 digits format. END_MONTH and END_YEAR should be presented together.
END_YEAR	Number of year to stop reporting production. Default = 0, in 4 digits format. END_MONTH and END_YEAR should be presented together.
<BRANCH NAME=…>	The following branches can be specified to be included: /PRODUCTION_SET/PRODUCING_ENTITY/METADATA /PRODUCTION_SET/PRODUCING_ENTITY/HEADER /PRODUCTION_SET/PRODUCING_ENTITY/LOCATION /PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE /PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE/METADATA /PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE/HEADER /PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE/LOCATION /PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE/TESTS /PRODUCTION_SET/PRODUCING_ENTITY/PRODUCTION /PRODUCTION_SET/PRODUCING_ENTITY/PRODUCTION/ABSTRACT /PRODUCTION_SET/PRODUCING_ENTITY/PRODUCTION/YEAR/REPORTED

/PRODUCTION_SET/PRODUCING_ENTITY/PRODUCTION/YEAR/MONTH

/PRODUCTION_SET/PRODUCING_ENTITY/INJECTION

/PRODUCTION_SET/PRODUCING_ENTITY/INJECTION/ABSTRACT

/PRODUCTION_SET/PRODUCING_ENTITY/INJECTION/YEAR/REPORTED

/PRODUCTION_SET/PRODUCING_ENTITY/INJECTION/YEAR/MONTH

/PRODUCTION_SET/PRODUCING_ENTITY/CONTENT

Example:

```
<EXPORT>
     <TEXTUAL_EXPORTS>
          <PRODUCTION_XML START_MONTH='01' START_YEAR='2000' END_MONTH='12'
END_YEAR='2013'>
               <BRANCH NAME='/PRODUCTION_SET/PRODUCING_ENTITY/HEADER'/>
               <BRANCH NAME='/PRODUCTION_SET/PRODUCING_ENTITY/LOCATION'/>
               <BRANCH
NAME='/PRODUCTION_SET/PRODUCING_ENTITY/WELLBORE/TESTS'/>
               <BRANCH
NAME='/PRODUCTION_SET/PRODUCING_ENTITY/PRODUCTION/YEAR'/>
               <BRANCH
NAME='/PRODUCTION_SET/PRODUCING_ENTITY/INJECTION/YEAR'/>
          </PRODUCTION_XML>
     </TEXTUAL_EXPORTS>
</EXPORT>
```

WELL_WORKBOOK	
	Attribute:
FORMAT=	Possible Values are XLSX, XLS and CSV.
	Elements:
<EXPORT_TABLE NAME=…>	The following “tabs” or sheets are available in the Excel spreadsheet. The order of the tables in the request indicate the order of the tabs in the actual Excel spreadsheet. WellboreHeaderTabularExport, WellboreLocationTabularExport, WellboreLocationNarrTabularExport, WellboreFormationTabularExport,

WELL_WORKBOOK
WellboreFaultTabularExport,
WellboreTestTabularExport,
WellboreTestTreatmentTabularExport,
WellboreTreatmentSummaryTabularExport,
WellboreTestPerforationTabularExport,
WellboreTestNarrativeTabularExport,
WellboreDwightsTestNarrTabularExport,
WellboreDstTabularExport,
WellboreDstPipeRecTabularExport,
WellboreDstMaterialTabularExport,
WellboreDstNarrativeTabularExport,
WellboreCoresTabularExport,
WellboreCoreDescriptionTabularExport,
WellboreCoreNarrativeTabularExport,
WellboreLogTabularExport,
WellboreMudTabularExport,
WellboreStringTabularExport,
WellboreDrillingJournalTabularExport
WellboreDrillingNarrTabularExport,
WellboreSurveyTabularExport,
WellboreSurveyPointTabularExport
WellboreCompletionTabularExport
WellborePressureTabularExport
WellboreLLSourceTabularExport

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
            <WELL_WORKBOOK FORMAT='XLSX'>
                  <EXPORT_TABLE NAME='WellboreHeaderTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreLocationTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreLocationNarrTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreFormationTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreFaultTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreTestTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreTestTreatmentTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreTreatmentSummaryTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreTestPerforationTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreTestNarrativeTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDwightsTestNarrTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDstTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDstPipeRecTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDstMaterialTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDstNarrativeTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreCoresTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreCoreDescriptionTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreCoreNarrativeTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreLogTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreMudTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreStringTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDrillingJournalTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreDrillingNarrTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreSurveyTabularExport'/>
                  <EXPORT_TABLE NAME='WellboreSurveyPointTabularExport'/>
                  <EXPORT TABLE NAME='WellboreCompletionTabularExport'/>
                  <EXPORT TABLE NAME='WellborePressureTabularExport'/>
                  <EXPORT TABLE NAME='WellboreLLSourceTabularExport'/>
            </WELL_WORKBOOK>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

PRODUCTION_WORKBOOK

	Attribute:
FORMAT=	Possible Values are XLSX, XLS and CSV.
START_MONTH	Number of month to start reporting production. Default = 0. In 2 digits format. START_MONTH and START_YEAR should be presented together.
START_YEAR	Number of year to start reporting production. Default = 0. In 4 digits format. START_MONTH and START_YEAR should be presented together.
END_MONTH	Number of month to stop reporting production. Default = 0. In 2 digits format. END_MONTH and END_YEAR should be presented together.
END_YEAR	Number of year to stop reporting production. Default = 0. In 4 digits format. END_MONTH and END_YEAR should be presented together.
	Elements:
<EXPORT_TABLE NAME=…>	The following 'tabs' or tables are available in the Excel spreadsheet. The order of the tables in the request indicate the order of the tabs in the actual Excel spreadsheet.
	ProductionHeaderTabularExport,
	ProductionWellTabularExport,
	ProductionTestTabularExport,
	ProductionAbstractTabularExport,
	AnnualProductionTabularExport,
	MonthlyProductionTabularExport,
	InjectionAbstractTabularExport,
	AnnualInjectionTabularExport,
	MonthlyInjectionTabularExport
	CO2AbstractTabularExport,
	AnnualCO2TabularExport,
	MonthlyCO2TabularExport
	IPCumNormTabularExport

Example:

```
<EXPORT>
      <TEXTUAL_EXPORTS>
            <PRODUCTION_WORKBOOK START_MONTH='01' START_YEAR='2000'
END_MONTH='12' END_YEAR='2006'>
                  <EXPORT_TABLE NAME='ProductionHeaderTabularExport'/>
                  <EXPORT_TABLE NAME='ProductionWellTabularExport'/>
                  <EXPORT_TABLE NAME='ProductionTestTabularExport'/>
                  <EXPORT_TABLE NAME='ProductionAbstractTabularExport'/>
                  <EXPORT_TABLE NAME='AnnualProductionTabularExport'/>
                  <EXPORT_TABLE NAME='MonthlyProductionTabularExport'/>
                  <EXPORT_TABLE NAME='IPCumNormTabularExport'/>
            </PRODUCTION_WORKBOOK>
      </TEXTUAL_EXPORTS>
</EXPORT>
```

Chapter Eight

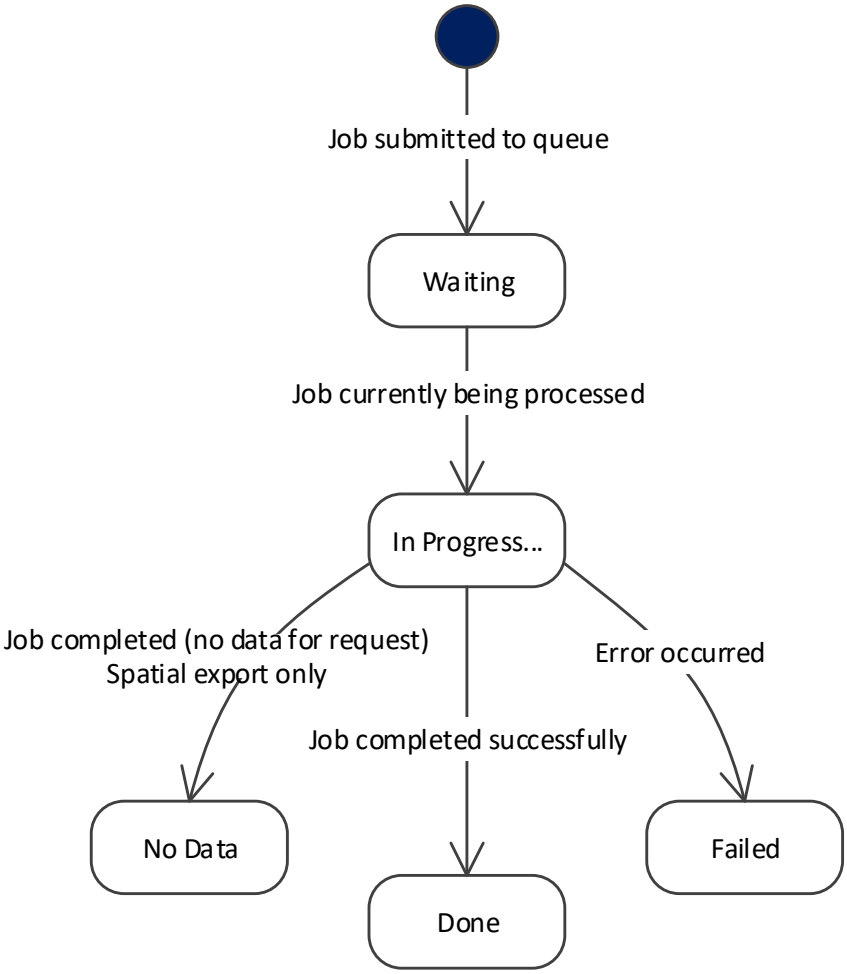
Appendix B: Oneline Data Template Schema

DataTemplate.xsd

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:element name='custom_oneline'>
          <xs:complexType>
               <xs:sequence>
                    <xs:element name='attribute' type='xs:string'
maxOccurs='unbounded'/>
               </xs:sequence>
               <xs:attribute name='domain' type='xs:string' use='required'/>
               <xs:attribute name='title' type='xs:string' use='required'/>
               <xs:attribute name='description' type='xs:string'
use='optional'/>
               <xs:attribute name='datatype' use='required'>
                    <xs:simpleType>
                         <xs:restriction base='xs:string'>
                              <xs:enumeration value='well'/>
                              <xs:enumeration value='Well'/>
                              <xs:enumeration value='production'/>
                              <xs:enumeration value='Production'/>
                         </xs:restriction>
                    </xs:simpleType>
               </xs:attribute>
               <xs:attribute name='createdBy' type='xs:string'
use='optional'/>
               <xs:attribute name='type' type='xs:string' use='optional'/>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

Appendix: Lifecycle of Job / Status Codes

Export, Graph and Report Jobs have the following life cycle.



Chapter Eight

Appendix C: QueryBuilder LookupXML Schema

LookupXML.xsd

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
      <xs:element name='ATTRIBUTE_SET'>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref='ATTRIBUTE' maxOccurs='unbounded'/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name='ATTRIBUTE'>
            <xs:complexType>
                  <xs:choice>
                        <xs:element ref='NAMECODE_RECORD' minOccurs='0'
maxOccurs='unbounded'/>
                        <xs:element ref='NAME_RECORD' minOccurs='0'
maxOccurs='unbounded'/>
                        <xs:element ref='CODE_RECORD' minOccurs='0'
maxOccurs='unbounded'/>
                        <xs:element ref='CODETYPE_RECORD' minOccurs='0'
maxOccurs='unbounded'/>
                  </xs:choice>
                  <xs:attribute name='NAME' type='xs:string' use='required'/>
            </xs:complexType>
      </xs:element>
      <xs:element name='NAME_RECORD'>
            <xs:complexType>
                  <xs:attribute name='NAME' type='xs:string' use='required'/>
                  <xs:attribute name='COUNT' type='xs:integer' use='required'/>
            </xs:complexType>
      </xs:element>
      <xs:element name='CODE_RECORD'>
            <xs:complexType>
                  <xs:attribute name='CODE' type='xs:string' use='required'/>
                  <xs:attribute name='COUNT' type='xs:integer' use='required'/>
            </xs:complexType>
      </xs:element>
      <xs:element name='NAMECODE_RECORD'>
            <xs:complexType>
                  <xs:attribute name='NAME' type='xs:string' use='required'/>
                  <xs:attribute name='CODE' type='xs:string' use='required'/>
                  <xs:attribute name='COUNT' type='xs:integer' use='required'/>
            </xs:complexType>
      </xs:element>
      <xs:element name='CODETYPE_RECORD'>
            <xs:complexType>
                  <xs:attribute name='CODE' type='xs:string' use='required'/>
                  <xs:attribute name='TYPE' type='xs:string' use='required'/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Chapter Eight

Appendix D: Common Errors

Entitlements

If a user is entitled to the data type but lacks entitlements for any entities in the queried region, the message is as follows, and the requested output is not created:
Failed to build an export: com.ihsenergy.enerdeq.exportbuilder.ExportBuilderException: No ids to export.

If a user asks for a datatype for which they are not licensed, the error message is:
Failed to build an export: com.ihsenergy.enerdeq.exportbuilder.ExportBuilderException: idTable cannot be null or empty

If the user specify a zero count query (due to entitlements or invalid IDs), the error message is:
System.web.services.ProtocolsSoapException: No ids to export

If map layers are not entitled, an empty export file is generated.

Template

Misspelled template name, the error message is:
System.web.services.Protocols.SoapException: Invalid Export Misspelled template name,

Template name and type of IDs mismatch, the error message is:
Exception thrown: System.Web.Services.Protocols.SoapException: Failed to build the job
Or
Exception thrown: System.web.services.Protocols.SoapException: No ids to export

No template name, the error message is:
Error: template not specified

URL

Invalid URL, the error message is:
System.net.WebException: the underlying connection was closed: The remote name could not be resolved

Authentication

Correct username, invalid password, the error message is:
System.Web.Services.protocols.SoapException: Authentication Failed: Invalid Username or password. Please try again.

Invalid username, correct password, the error message is:
System.Web.Services.protocols.SoapException: Authentication Failed: Invalid Username or password. Please try again.

Good username, no password, the error message is:
Password not specified

Application name

Application name is null, the error message is:
Error: calling application not specified

Domain

Invalid domain name, the error message is:
System.web.services.ProtocolsSoapException: Invalid Domain

Output filename

Disallowed symbols in filename, the error message is:
SOAP exception: invalid filename

Null filename, the error message is:
Error: filename not specified

Empty spatial export file

Map layers are not entitled.

Map layers are not within the requested Lat/Long coordinates.

Temporary capacity errors

File is too big, the error message is:
SOAP exception: We are currently unable to process your current request for <x> entities due to the overall size of the request. Please reduce the size of your request and re-submit at your earliest convenience --Thank you.

Request Limits to the following methods

ExportBuilder Build():100,000 well or production IDs returned by the query. The limit will apply to all export templates except the Well ID List and Production ID List. There is no limit when using the standard Well ID List or Production ID List templates..

ExportBuilder BuildFromQuery(): 100,000 well or production IDs returned by the query. The limit will apply to all export templates except the Well ID List and Production ID List. There is no limit when using the standard Well ID List or Production ID List templates.

ExportBuilder BuildOneline(): 100,000 well or production IDs. The limit will apply to both standard and custom Oneline templates.

ExportBuilder BuildOnelineFromQuery(): 100,000 well or production IDs returned by the query. The limit will apply to both standard and custom Oneline templates.

GraphBuilder Build(): 500 well or production IDs. The limit will apply to all Graph templates.

GraphBuilder BuildFromQuery(): 500 well or production IDs returned by the query. The limit will apply to all Graph templates.

ReportBuilder Build(): 10,000 well or production IDs. The limit will apply to all Report templates.

ReportBuilder BuildFromQuery(): 10,000 well or production IDs. The limit will apply to all Report templates.

Chapter Eight

Appendix E: ChangeDelete Schema & Sample

Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
      <xs:element name='ChangeDeleteRecords'>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref='ChangeDelete' maxOccurs='unbounded'/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name='ChangeDelete'>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name='Uwi' type='xs:string'/>
                        <xs:element name='Source' type='xs:string'/>
                        <xs:element name='Sequence' type='xs:int'/>
                        <xs:element name='Date' type='xs:string'/>
                        <xs:element name='NewUwi' type='xs:string'/>
                        <xs:element name='ReferenceUwi' type='xs:string'/>
                        <xs:element name='Remark' type='xs:string'/>
                        <xs:element name='ReasonCode' type='xs:string'/>
                        <xs:element name='ActiveCode' type='xs:string'/>
                        <xs:element name='Proprietary' type='xs:string'/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Sample

```
<?xml version='1.0' encoding='UTF-8'?>
<ChangeDeleteRecords>
  <ChangeDelete>
    <Uwi>12081794622010</Uwi>
    <Source>PI</Source>
    <Sequence>4620503</Sequence>
    <Date>5/3/2010 16:57:1</Date>
    <NewUwi>12023794622010</NewUwi>
    <ReferenceUwi />
    <Remark>CCC - CHANGE IC ON WELL UNDER IC CONTROL  (ADD)</Remark>
    <ReasonCode>9</ReasonCode>
    <ActiveCode />
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>09021202030100</Uwi>
    <Source>PI</Source>
    <Sequence>4921687</Sequence>
    <Date>8/27/2010 17:9:7</Date>
```

```
    <NewUwi />
    <ReferenceUwi>09021202030100</ReferenceUwi>
    <Remark>DEL - WELL IS BEING READDED WITH NEW OR SAME API</Remark>
    <ReasonCode>4</ReasonCode>
    <ActiveCode>N</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>42235747922010</Uwi>
    <Source>PI</Source>
    <Sequence>4955929</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi />
    <ReferenceUwi />
    <Remark>DEL - NO SUCH WELL EXISTS</Remark>
    <ReasonCode>3</ReasonCode>
    <ActiveCode>Y</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>42503747932010</Uwi>
    <Source>PI</Source>
    <Sequence>4955963</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi>42503417590000</NewUwi>
    <ReferenceUwi />
    <Remark>CCC - CHANGE IC ON WELL UNDER API CONTROL (ADD)</Remark>
    <ReasonCode>8</ReasonCode>
    <ActiveCode>Y</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>12191000750001</Uwi>
    <Source>PI</Source>
    <Sequence>4955967</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi>12191000750000</NewUwi>
    <ReferenceUwi />
    <Remark>CCC - API NUMBER CHANGE MADE BY REGULATORY AGENCY</Remark>
    <ReasonCode>5</ReasonCode>
    <ActiveCode>Y</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>23089200830002</Uwi>
    <Source>PI</Source>
    <Sequence>4955969</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi>23089200830000</NewUwi>
    <ReferenceUwi />
    <Remark>CCC - CONTROL CODE CHANGE SHOULD NOT GO TO WHCS</Remark>
    <ReasonCode>7</ReasonCode>
```

```
    <ActiveCode />
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>42131380910001</Uwi>
    <Source>PI</Source>
    <Sequence>4955970</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi />
    <ReferenceUwi>42131380910000</ReferenceUwi>
    <Remark>DEL - WELL IS DUPLICATED UNDER ANOTHER API NUMBER</Remark>
    <ReasonCode>2</ReasonCode>
    <ActiveCode>Y</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
  <ChangeDelete>
    <Uwi>42195002330000</Uwi>
    <Source>PI</Source>
    <Sequence>4955971</Sequence>
    <Date>9/2/2010 17:21:1</Date>
    <NewUwi>42195002330101</NewUwi>
    <ReferenceUwi />
    <Remark>CCC - WELL WAS ENTERED UNDER INCORRECT API NUMBER</Remark>
    <ReasonCode>1</ReasonCode>
    <ActiveCode>Y</ActiveCode>
    <Proprietary>N</Proprietary>
  </ChangeDelete>
</ChangeDeleteRecords>
```

IHS Markit Customer Care:

CustomerCare@ihsmarkit.com
CustomerCareEnergyWebServices@ihsmarkit.com
Americas: +1 800 IHS CARE (+1 800 447 2273)
Europe, Middle East, and Africa: +44 (0) 1344 328 300
Asia and the Pacific Rim: +604 291 3600

IHS Markit™