Demand Resource (DR)

User Interface Guide

Document Version: 2.0

Posted Date: Feb 27, 2018

Contents

Table of Figures

About This Document

This document describes the usage of the Demand Resource (DR) User Interface (MUI) from the perspective of the end user, such as a Demand Designated Entity (DDE), meter reader, market participant, or ISO New England Customer Support staff. This document *does not* describe the MUI from the perspective of Information Technology (IT) administrators at ISO New England wishing to install and configure MUI instances.

Purpose of This Document

This document should serve as the primary documentation for the end users of the MUI. This is a guide for both the graphical and programmatic (web services) components of the MUI for the Demand Resource Integration component of the ISO New England energy market.

Who Should Use This Document

DDEs, meter readers, market participants and other users wishing to use MUI functionality should use this document. An assumption has been made that the reader of this user guide is familiar with using web-based applications and web-browser navigation. The user of the DR MUI should have a good understanding of the ISO New England Transmission, Markets & Services Tariff, ISO New England Manuals and other governing documents to properly use the DR MUI to complete their desired business intentions.

Structure of This Document

This document contains the following major sections:

- Chapter 1 is a brief introduction to the MUI.
- Chapter 2 describes the usage of the MUI graphical interface. This includes setting up a web browser to connect to the MUI, the common functions of the MUI, as well as a detailed description of the behavior and usage of each display.
- Chapter 3 describes the usage of the MUI programmatic interface. This includes general concepts and conventions used in the MUI web service, instructions to build, configure and run the provided sample programs, as well as a detailed description of each service operation, including its format and behavior. This chapter is meant to serve as a supplement to the WSDL and provided sample programs for the service.

More Information

In terms of the programmatic interface to the MUI, this document is meant to serve as a supplement to the web service definition language (WSDL) for the MUI web service. In addition to the WSDL, two sample programs are available (one written in Java, the other in .NET) to illustrate examples of how to code against the web service application programming interface (API) provided by the MUI.

Conventions

This document contains various references to example XML (Extensible Markup Language), expressed in grey blocks. For simplicity, these examples do not always show fully valid XML, because it may not be practical or necessary to show valid XML to illustrate the full functionality of a given message. In addition, the XML always assumes that all items are in the default namespace, and that the body of a SOAP (Simple Object Access Protocol) envelope is wrapped around the message. This of course is not really accurate in terms of an actual web service message that should be provided to the service. For instance, the following example:

```
<SubmitTelemetryCorrections  party="foo" day=”2018-01-01”>…
```

Would really be expressed as:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:mui="http://www.markets.iso-ne.com/MUI/DR/Messages">
   <soap:Header/>
   <soap:Body>
     <mui: SubmitTelemetryCorrections  party="foo" day=”2018-01-
01”/>…
   </soap:Body>
</soap:Envelope>
```

Change Summary

Document Revision	Date	Comments
1.0	April 14, 2010	Edited for release and posting based on E1071.7 [ghl]
RL.1	May 25, 2010	Added information on submittal validation errors for Hourly Availability, Maximum Hourly Demand Reduction, and Telemetry Corrections. Fixed typo with “genMW” in section 4.3.7.1. Clarified that “loadMW” value may be positive or negative in section 4.3.7.1. Merged all changes made by ISO-NE as sent to Areva on May 25, 2010 [rgg]
RL.2	July 18, 2011	Edited to reflect change from use of xsd:time to xsd:dateTime [rgg]
2.0	Feb 27, 2018	Edited to reflect PRD changes [rgg]

1. Introduction

The DR MUI is a web-based system that allows market participants and other users to interact with the energy market over the public internet in a secure manner. The MUI provides two mechanisms to do this:

- A graphical user interface (GUI) that runs in a web browser. Please refer to the **Web Browser Support for SMD Applications** page on the ISO New England website, which can be found under Participate > Support > Web Browser Support, Web Browser Support for SMD Applications section of our website (which can be found under Participate > Support > Web Browser Support) for information on the browsers and security protocols ISO New England supports.
- A programmatic interface using web services. Users can write software against this interface to integrate the MUI functionality with their own systems.

Authentication to the interface is done using client certificates issued by ISO New England. Potential users need to register with the Customer Support Department within ISO New England in order to obtain valid certification for access to the MUI. Once registered, users will be provided with the link to access the SMD Application Home Page which will then allow direct access to the DR MUI software application. Questions or inquires about certifications for DR MUI access should be addressed to the Customer Support Department at ISO New England

These client certificates are used by the MUI to facilitate two-way encryption in SSL connections with the MUI servers.

This document discusses the use of both of these interfaces, as it is likely that many users will use either in different contexts or for different purposes. The document is broken into these two chapters:

- The chapter on the GUI describes how to connect your web browser to the MUI, as well as navigation and basic use of the displays.
- The chapter on the web service is meant to serve as a supplement to the provided WSDL and sample programs for the web service, and will illustrate the message format and behavior of each web service operation. This chapter of the document also provides instructions for building, configuring and running the provided sample programs in both Java and .NET.

2. User Roles and Market Agents

As a user of the MUI, you will be granted one or more of the following roles in the ISO New England Customer Asset Management System (CAMS) by your Security Administor (SA):

- **DDE Read Only** has permission to query, asset interval data (including baselines), resource interval data and asset telemetry as a DDE.
- **DDE Read Write** has all the permissions of DDE Read Only plus permission to submit telemetry corrections as a DDE.
- **DR Lead Part Read Only** has all the same permissions as DDE Read Only, except that this user views resources and assets they own instead of those for which they are the DDE.
- **DR Lead Part Read Write** has all the same permissions as DDE Read Write, except that this user views resources and assets they own instead of those for which they are the DDE.
- **Meter Reader Read Only** has permissions to query asset telemetry and asset interval data for assets where they are the meter reader.
- **Meter Reader Read Write** has the permissions of Meter Reader Read Only plus permission to submit telemetry corrections for assets where they are the meter reader.

Each role applies to a logical participant in the market. Therefore, as a market agent (someone who represents another market participant or participants), you will be granted a role corresponding with your relationship to that user. Consider the following example:

There are two participants, A & B, and the system has assets 1 through 4. The system models the relationship between the participants and these assets in the following table:

Participant	Asset	Relationship
A	1	DDE
A	2	DDE
B	3	DDE
B	4	MeterReader

Assume then that you have the role DDE Read Write for participant A, and Meter Reader Read Write for participant B. This has the following consequences:

- You may submit telemetry and the rest of the DDE functions for assets 1 and 2, since you have DDE Read Write for A and A is the DDE for those assets.
- You may submit telemetry corrections for asset 4, since you have Meter Reader Read Write for B.
- You may NOT execute any of the DDE functions for either asset 3 or 4, because you do not have a DDE relationship with participant B, even though B may be the DDE for asset 3.

In the MUI, you may only operate as one participant at a time. With the graphical interface, you select the participant you wish to act as. From the web service interface, you must specify the participant you will act as for a given transaction.

3. Using the Graphical Interface

The MUI graphical interface is a web application that will run in Mozilla Firefox and Microsoft Internet Explorer. Along with a digital certificate provided by ISO New England, one of these browsers will be needed to access the MUI. In order for the MUI servers to identify you, the valid certificate from ISO New England must be installed in one of the supported web browsers. The instructions to do this differ between web browsers. Once this is done, the MUI will automatically authenticate when you log in.

3.1 Common Display Functions

The following section describes the basic functions of the MUI, including functions that are common across displays.

3.1.1 Selecting a Participant

If you have more than one participant you represent, you will be presented with a drop-down list of the participants you may represent. Until you select a participant, all the displays in the MUI will show no data. If you represent only one participant, the drop-down will not be shown and you will simply always represent that participant:

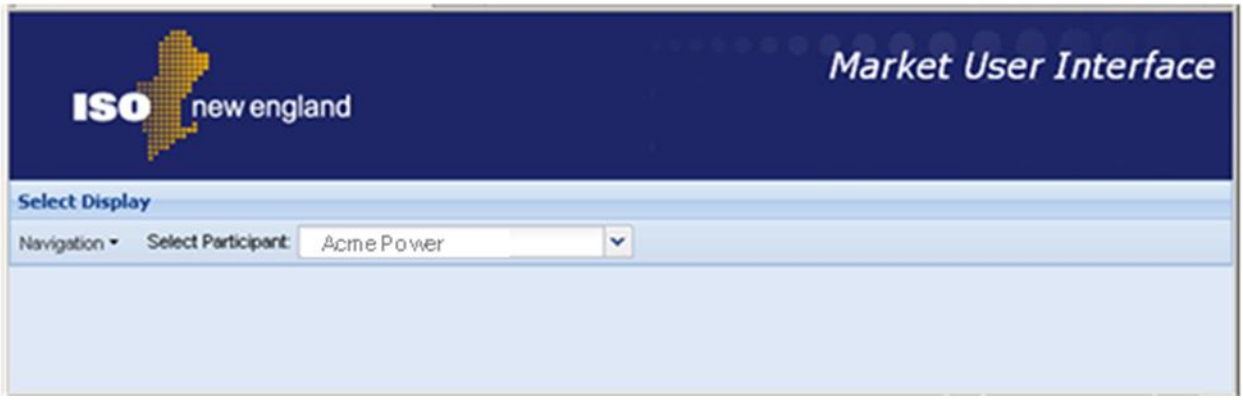

Figure 1: Select Participant Drop-down

If you select a participant while a display is open, that display will be refreshed to reflect your new context.

3.1.2 Navigation

After navigating to the MUI URL, you will be presented with the following blank screen:

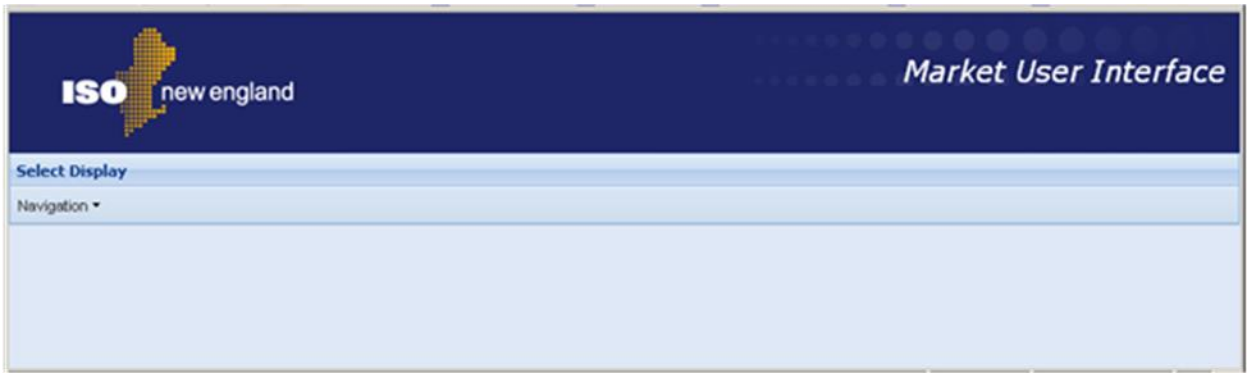

Figure 2: Initial Login

Each display in the application can be found under the **Navigation** menu. After clicking on one of the items in this menu, the display will be shown in the bottom pane.

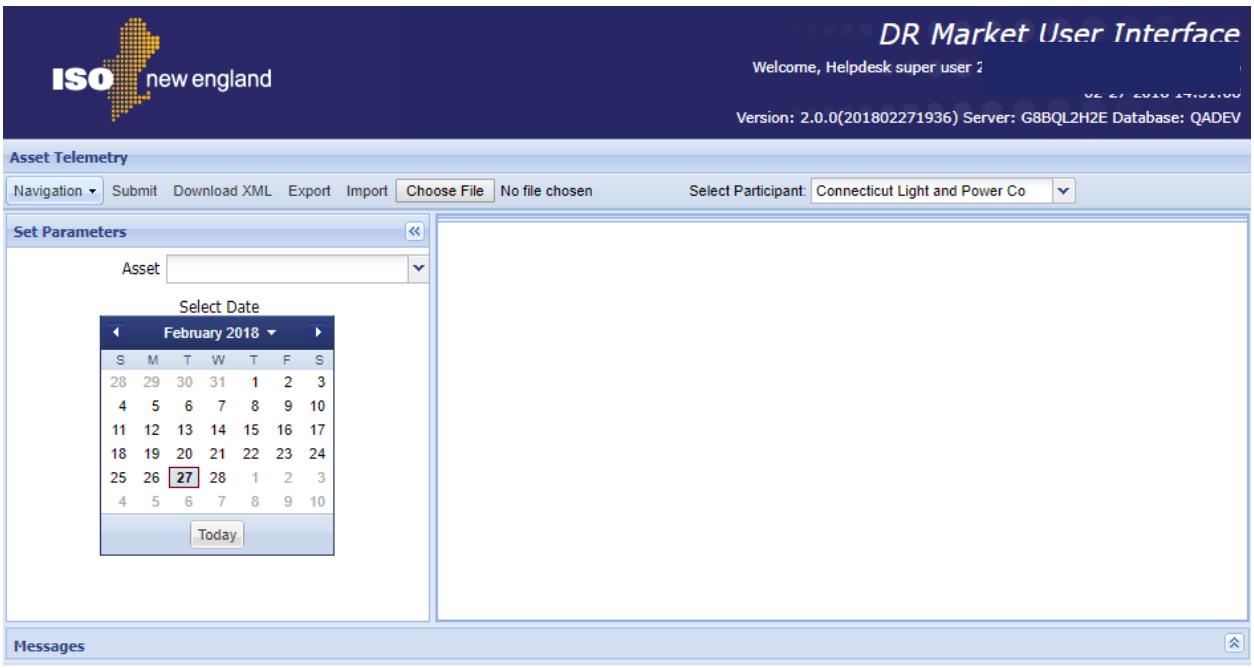

Figure 3: Basic Display Layout

Any changes you make on this display can be saved by clicking on the **Submit** button. To navigate to another display, use the **Navigation** menu.

WARNING: When you navigate from one display to another, any information that you have not saved from the previous display will be lost.

3.1.3 XML Import and Export Functions

Each display includes a capacity to import and export the state of the screen to the same XML formats used by the web service interface[1]. Do this with the tools provided on the toolbars:

- **Export** will download the current deltas of the screen as XML.
- **Import** will import an XML file and load it as deltas on the current display. Before running this function, click the **Browse…** button on the toolbar to select a file to upload.
- **Download XML** will run the same query as the currently highlighted tab on the UI as if it came through the web service API. An XML document will be returned that includes the same SOAP envelope that would have been included in the query response.

3.1.4 Uploading XML

Some users may wish to create a partially automated interface, where instead of invoking the web service directly; the user can generate the XML as files and upload them manually through the graphical interface. This is done using the **Upload XML** display, available to every user.

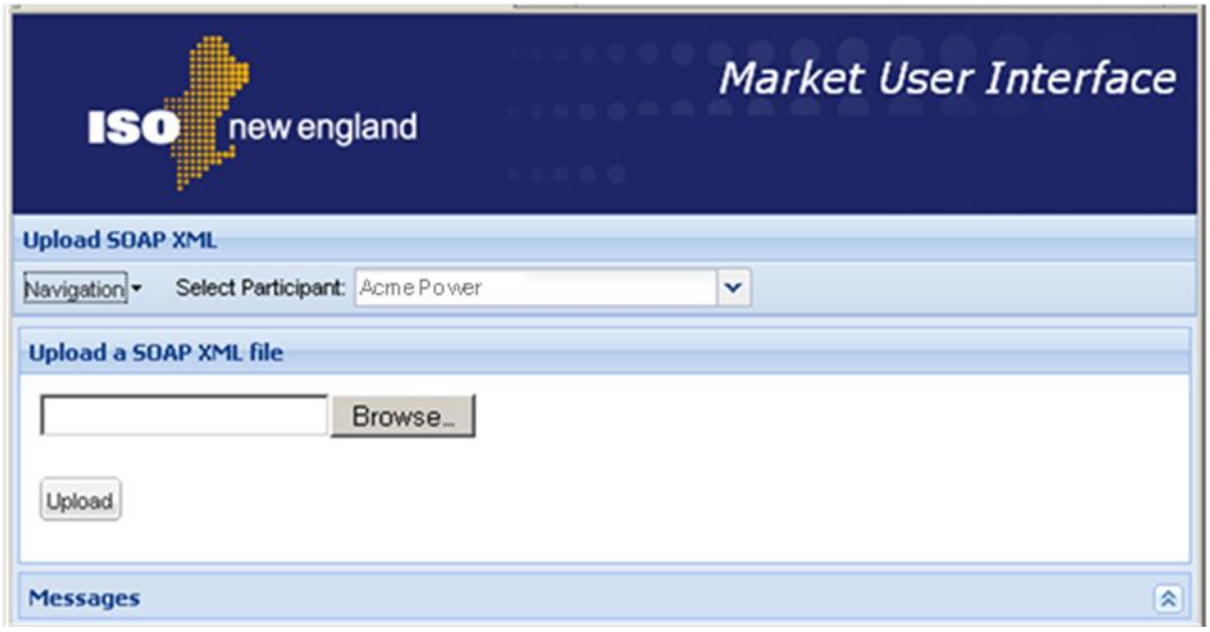

Figure 4: Upload XML Display

This display will allow you to upload files to the MUI. The format and syntax for these is exactly the same as it is in the web service API. After

[1] This does not include wrapping in a SOAP envelope.

sending a file, you will be given the exact same result back as an XML document that you would have received if you called the web service.

WARNING: For the sake of consistency, the upload screen will not use the participant you currently have selected. The uploads will adhere to the "party" attribute in the XML document, just as the web service API does. Make sure your XML documents are syntactically correct.

3.1.5 Validation Errors

The MUI submissions can be viewed as "all or nothing", in that all data will go through if there are no errors, or no data will go through if there are errors. After submission, errors will appear in red in the **Messages** pane at the bottom of the screen.

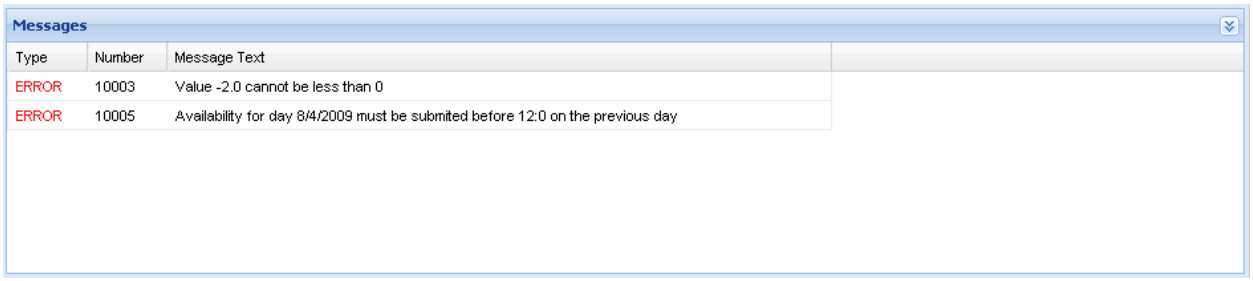

Figure 5: Errors Pane

If any errors occur here, then **No Data was Submitted**. Correct the errors and click **Submit** to resubmit your data.

3.2 Demand Response Displays

The following section describes the specific display functions for demand resources.

3.2.1 Asset Interval Data

DDEs and Meter Readers can use the Asset Interval Data display to view baseline and performance data in the BLTS system for a selected asset for all 5-minute intervals of the selected day. Based on the selected asset ID and selected day, the data grid may contain several empty fields or no data.

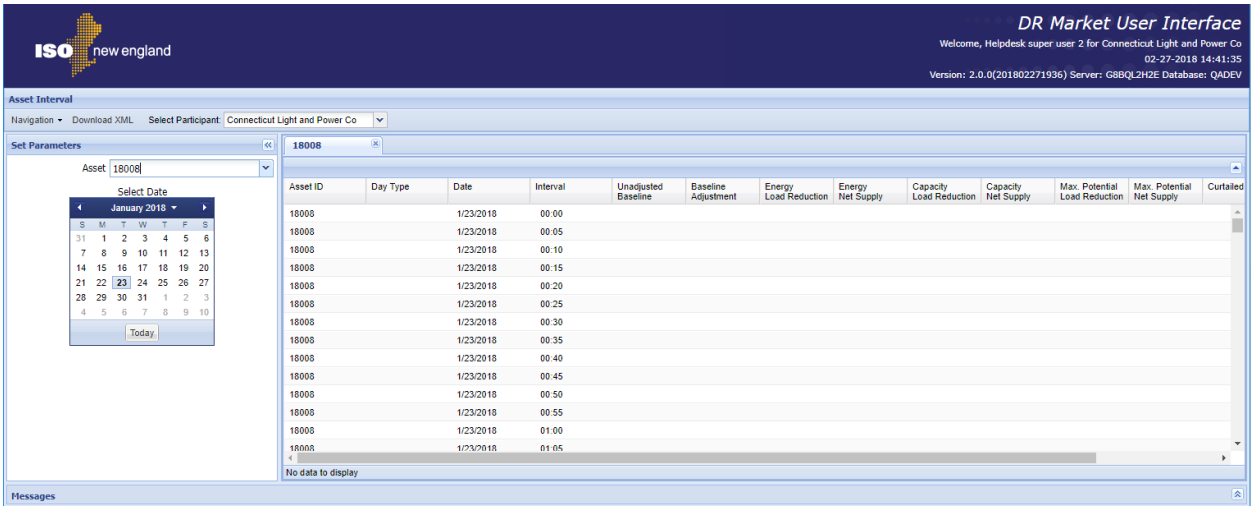

Figure 6: Asset Interval Data Display

To open asset interval data for a particular day and asset, first select the desired day if it is not already selected, then select the asset. A tab will be opened displaying the current asset interval values.

3.2.2 Asset Telemetry

DDEs and Meter Readers can use the Asset Telemetry display to view the current telemetry in the BLTS system, as well as make corrections to the telemetry stored there.

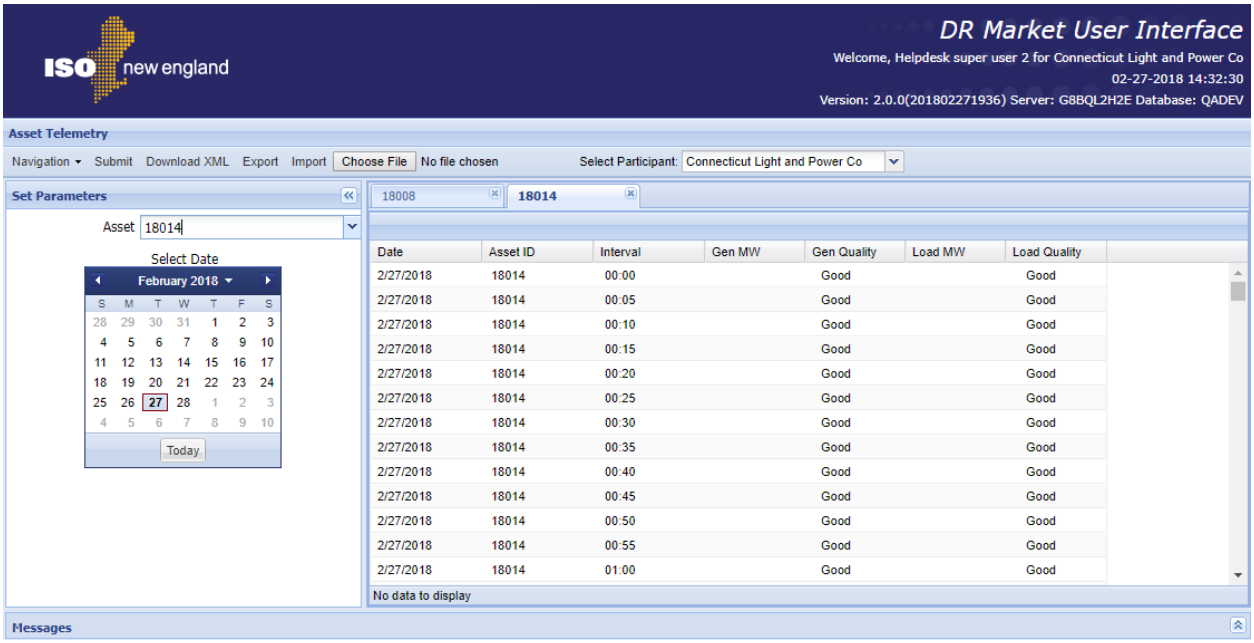

Figure 7: Asset Telemetry Display

To open telemetry for a particular day and asset, first select the desired day if it is not already selected, then select the asset. A tab will be opened displaying the current telemetry values. To edit the values, click on the desired cell in the grid and enter the new value. Any corrections made through the MUI will be displayed in blue. To save the data, click the **Submit** button.

3.2.3 Resource Interval Data

DDEs and Meter Readers can use the Resource Interval Data display to view the performance data in the BLTS system for a selected resource for all 5-minute intervals of the selected day. Based on the selected resource ID and selected day, the data grid may contain several empty fields or no data.

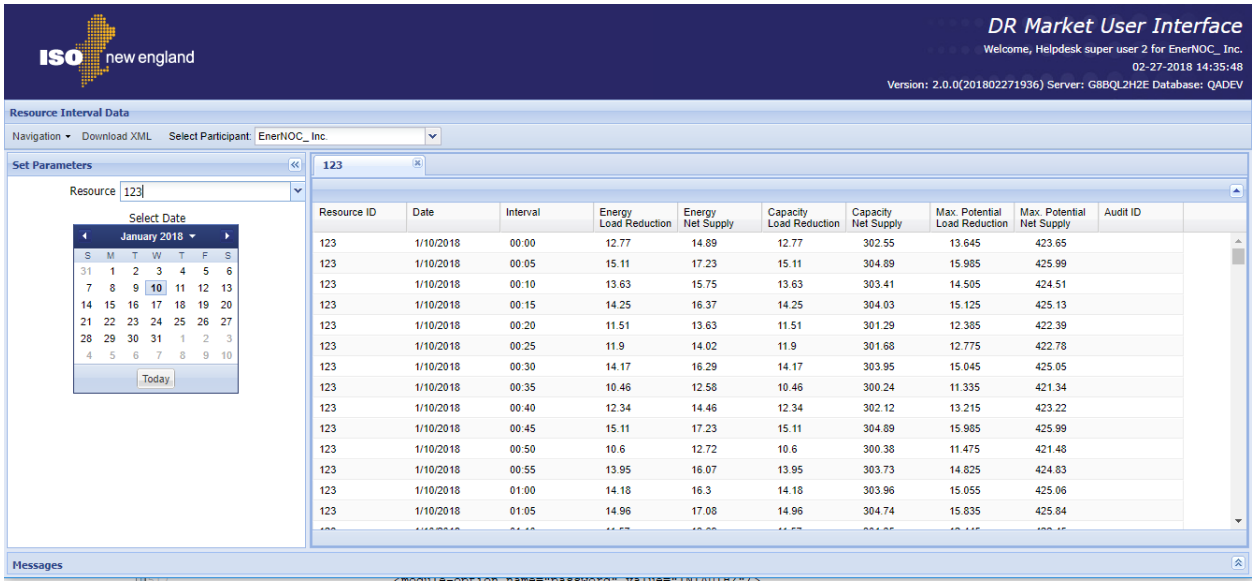

Figure 8: Resource Interval Data Display

To open resource interval data for a particular day and resource, first select the desired day if it is not already selected, then select the resource. A tab will be opened displaying the current resource interval data values.

4. Using the Programmatic Interface

The programmatic interface to the MUI is a standard SOAP 1.2/WSDL 1.1 compliant web service. As a web service, the MUI API is secure, scalable and cross-platform; a client application can be written in nearly any modern enterprise technology and language, such as Java, .NET, C++, Ruby, PhP etc. This section of the document describes the general use of the MUI web service in particular, as well as each of the specific functions. This document **DOES NOT** describe the concepts of web services in general. For a good reference on web service concepts, visit http://www.w3schools.com/webservices/default.asp. This document is also a supplement to several additional artifacts, including:

- WSDL and referenced XSDs.
- A sample client program written in Java
- Another sample client program written in C# (.NET).

This document includes setup instructions for both of these sample programs.

4.1 General Web Service Concepts

All web service operations in the MUI fall into two distinct categories: queries and submittals. As a general rule, all submittal messages have a corresponding query, such that the user may query data that he has submitted.

If any errors occur during a query or submission, a fault will be returned that includes a list of errors consisting of a numeric ID and an error message:

```
<MUIFault>
  <Error number="1">
    Some error message
  </Error>
  <Error number="2">
    Some other error message
  </Error>
</MUIFault>
```

If an error occurs during a submission, then none of the data submitted in the original message will be saved.

The successful response to query messages is different for each query, and is described in more detail below. The responses to successful submittals however are all the same, and consist of a transaction ID for reference as well as an optional set of warnings on the user's submission:

```
<SubmitConfirmation
    transactionId="dc523616-511b-4498-93b3-0371ec63cf25">
  <Warning number="1">
    <ErrorMessage>Some warning</ErrorMessage>
  </Warning>
</SubmitConfirmation>
```

4.1.1 Handling Times

Throughout the XML schema, dateTimes are used in various places. This is a standard schema type, and can therefore be in any time zone provided that the time zone offset is specified in the standard way (as "Z" or "-00:00" for GMT, or "-04:00" and "-05:00" for eastern daylight and eastern standard respectively). This is good practice and all samples in documentation will show timestamps referencing the Eastern Time rules, to mirror the market.

However, the standard does not require the use of a time zone offset. If no time zone is provided, the timestamp is assumed to be in local time. This can lead to ambiguities involving daylight savings transitions, so it is recommended that a time zone offset specifier is always used.

4.1.2 Market Agent Functionality

If you are a market agent that represents multiple participants, you must specify the participant you represent in each message. Each root submittal and query element contains a **party** attribute that should be set to the name of the participant you would like to execute the transaction as.

4.1.3 Validation and Error Handling

Each transaction sent to the MUI is an "all or nothing" event. The MUI will only commit data if no errors occur. If an error occurs, a fault will be sent to the user with the appropriate error messages, and **none of the data sent will be committed**. After correcting errors, be sure to resubmit the entire message.

4.1.3.1 Submission Windows

Some of the transactions have some specific validation rules regarding a "window" in which the data must be submitted. The time against which these validations are done will be the time that the MUI server receives the message for processing, which is the earliest time that the MUI server can recognize a valid document. Therefore, in the case of large transactions where processing may take up to several minutes, the user is not penalized for not taking that processing time into account.

4.2 Using the Sample Programs

Most modern enterprise technology stacks can integrate with web services. In supplement to this document, two sample client programs are included with some of the most popular tools: one written in Java, the other in .NET. This section describes the setup and architecture of these sample programs.

4.2.1 Java

The Java space perhaps contains the greatest variety of web service binding frameworks amongst enterprise technologies. A modern JAX-WS compatible stack is recommended, such as Apache CXF, GlassFish/Metro, JBossWS, Axis2 etc.

This sample uses Apache CXF to bind to the MUI web service. Apache CXF is an open source project hosted by the Apache Software Foundation. Documentation for CXF can be found at http://cxf.apache.org/.

4.2.1.1 Setup

The Java sample program requires the following pre-requisites to run:

- JDK 6. Be sure to add the JDK bin directory to the system path.
- Maven 2.0.9 is required to build the sample. To install Maven:
  - Unzip Maven
  - Create a new Environment Variable called M2_HOME with the path of the Maven installation
  - Add the maven bin directory to your system path

The program may also be run within Eclipse, with embedded Maven support installed. There is an Eclipse project file at the root directory of the sample.

To build the sample, unzip the program zip file. Next, run **BuildSample.bat** at the root of the directory you unzipped the program to. This will compile the Java classes, and build an executable JAR file.

4.2.1.1.1 Generating a Trust Store

You will need to generate a Java-formatted trust store with ISO New England's certificate authority or server certificates. The sample program will use this to determine what certificates to trust. To do this, the **keytool** command included with the JDK can be used. With the .crt file of the certificate, run the following command to generate a keystore:

keytool –keystore <whateveryouwanttocallit>.keystore –alias ca –import –file <yourcertificateauthority>.crt

You will be prompted for a password to the key store. After entering a password, this will generate the .keystore file needed in the next step.

4.2.1.2 Running the Program

To run the program, first edit /target/MUISample.properties underneath the root directory of the program. You need to change the following properties:

- client.key.file should point to your certificate file provided by ISO New England.
- client.key.pswd should be the password to your certificate.
- srvr.trust.kstr should be a Java key store containing the certificate authorities for ISO New England's certificates. This is required to tell your program what certificates to trust. This is discussed in more detail in the previous section.
- srvr.trust.pswd should be the password to the above key store.
- endpoint.link should be set to the URL of the MUI servers.

After editing this properties file, the program can be run by running **RunSample.bat** at the root of the directory you unzipped the program to.

4.2.1.3 Anatomy of the Program

Apache CXF / JAX-WS works by generating a plain Java proxy that internally marshals data to and from XML and sends it over secure HTTP to the MUI web servers. This allows the programmer to use the web service like any other Java API. Since the proxy is referred to through an interface, the web service can even be stubbed out in memory for unit testing.

The first step in doing this is generating the interfaces and classes that map to the MUI web service interface. The sample does this in the Maven pom.xml file using the CXF maven plugin. The following snippet from pom.xml shows the syntax of that configuration:

```
<!-- This config is for the apache CXF code generator.
    This will need to be run before the project will compile
    in Eclipse or any other IDE.  The plugin generates
    JAXB/JAX-WS bindings from the provided MUI WSDL
    file that can be called as simple Java objects.   -->
<plugin>
   <groupId>org.apache.cxf</groupId>
   <artifactId>cxf-codegen-plugin</artifactId>
   <version>2.1.4</version>
   <executions>
```

```
            <execution>
               <id>generate-sources</id>
               <phase>generate-sources</phase>
               <configuration>
               <sourceRoot>src/main/java</sourceRoot>
                  <wsdlOptions>
                     <wsdlOption>
                        <wsdl>src/main/resources/wsdl/DRMUI.wsdl</wsdl>
                        <extraargs>
                           <extraarg>-p</extraarg>
                           <extraarg>
                             com.areva.sample.jaxwsbindings
                           </extraarg>
                        </extraargs>
                     </wsdlOption>
                  </wsdlOptions>
               </configuration>
               <goals>
                  <goal>wsdl2java</goal>
               </goals>
            </execution>
         </executions>
      </plugin>
```

This generates code from the WSDL and XSD files and puts it into the **com.areva.sample.jaxwsbindings** package. Specifically, the interface **com.areva.sample.jaxwsbindings.DemandResourceMUI** can now be set up as a proxy to the web service. The sample client program is actually implemented in the **com.areva.sample.MUIClientSample** class under src/main/java. This code generation does not have to be done with Maven. The CXF Maven plugin is actually a wrapper for the command line utility **wsdl2java**, which ships with CXF.

The two most important methods in this class are those that configure the CXF client proxy. Alternatively, CXF integrates nicely with the Spring Framework, and all this configuration can be done through Spring XML.

The first method, **createWebService**, does most of the basic work of creating the client proxy:

```
/**   Create the CXF webservice Proxy.
 *    @return webService used throughout the project
 * */
private static DemandResourceMUI createWebservice(){
   // factory for creating interface
   JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();

   // logging for in and out
   factory.getInInterceptors().add(new LoggingInInterceptor());
   factory.getOutInterceptors().add(new LoggingOutInterceptor());

   // add interface and end point
   factory.setServiceClass(DemandResourceMUI.class);
   factory.setAddress(
         properties.getProperty(
```

```
                KeyName.ENDPOINT_LINK.toString()));

    // instance of our web service message options
    DemandResourceMUI webService = (DemandResourceMUI) factory.create();

    // add certificate connectivity properties
    configureSSLOnTheClient(webService);

    return webService;

}
```

This method creates an instance of the **DemandResourceMUI** interface that can be invoked like any other Java object. Since the MUI uses two-way SSL, the **configureSSLOnTheClient** method is also important in setting up the client. This method sets up the certificates that the connections will trust, and the certificates it will authenticate as:

```
/** This method configures our transport layer to use <i>two way</i>
      secure HTTP
 *    To do this we're configuring the certificates we trust as well as
      the certificate
 *    that we are.
 *    @param c - the CXF Proxy that we want to wrap with SSL
 *  */
private static void configureSSLOnTheClient(Object c){
    // end point obtained
    org.apache.cxf.endpoint.Client client = ClientProxy.getClient(c);

    // conduit we're using to transfer data
    HTTPConduit httpConduit = (HTTPConduit)client.getConduit();

    try{
       // holds parameters for configuration
       TLSClientParameters tlsParams = new TLSClientParameters();
       tlsParams.setDisableCNCheck(false); // don't disable

       // format type for server side certificates
       KeyStore trustStore = KeyStore.getInstance(
          properties.getProperty(KeyName.SRVR_TRUST_FMT.toString())
       );

       // location of server trusted list of certificates
       File serverTrusts = new File(
          properties.getProperty(KeyName.SRVR_TRUST_KSTR.toString())
       );

       // add trusted server certificates to store
       trustStore.load(new FileInputStream(serverTrusts),
         properties.getProperty(
          KeyName.SRVR_TRUST_PSWD.toString()).toCharArray()
       );

       // trust manager generated
       TrustManagerFactory trustFactory =
          TrustManagerFactory.getInstance(
                TrustManagerFactory.getDefaultAlgorithm()
```

```
        );// uses default Algorithm

      // initialize trust manager with the file acquired
      trustFactory.init(trustStore);

      // this configures our client to trust these certificates
      tlsParams.setTrustManagers(trustFactory.getTrustManagers());

      // keys for client side
      KeyStore keyStore = KeyStore.getInstance(
         properties.getProperty(KeyName.CLIENT_KEY_FMT.toString())
      );

      // location of client key
      File clientKey = new File(
         properties.getProperty(KeyName.CLIENT_KEY_FILE.toString())
      );

      // load certificate file into key store using password
      keyStore.load(new FileInputStream(clientKey),
         properties.getProperty(
           KeyName.CLIENT_KEY_PSWD.toString()).toCharArray()
      );

      // manager for keys
      KeyManagerFactory keyFactory =
         KeyManagerFactory.getInstance(
               KeyManagerFactory.getDefaultAlgorithm()
         );// uses default Algorithm

      // authorize use of key
      keyFactory.init(keyStore,
         properties.getProperty(
           KeyName.CLIENT_KEY_PSWD.toString()).toCharArray()
      );

      // this sets our client to use the configured key and certificate as
      // our credentials
      tlsParams.setKeyManagers(keyFactory.getKeyManagers());

      // add parameters to conduit
      httpConduit.setTlsClientParameters(tlsParams);

    // exceptions are caught and dealt with the same way
    } catch (Exception e) {
         System.out.println(
      "Security configuration failed with the following: ");
         e.printStackTrace();
         System.out.println("-------");
      }// end catch
  }//end
```

4.2.2 .NET

The .NET program uses Windows Communication Foundation (WCF), the standard framework in .NET for binding to web services. WCF works in a

way similar to Apache CXF in the Java example described above. WCF creates a simple .NET interface that can be configured as a proxy to the web service itself. Calling methods on the interface will marshal the parameter objects to XML and send them to the MUI web servers.

4.2.2.1 Setup

The .NET program uses all standard Microsoft tools. Everything required to build the program is installed with Microsoft Visual Studio 2008.

To build the program, unzip the program zip file. Then, open and build **MUISample.sln** under the root where the file was unzipped.

4.2.2.2 Running the Program

To run the program, first edit MUISample.properties in the directory where the executable is built. You need to change the following properties:

- client.key.file should point to your certificate file provided by ISO New England.
- client.key.pswd should be the password to your certificate.
- srvr.trust.kstr should be ISO New England's server or certificate authority certificate (.crt file).
- srvr.trust.pswd should be the password to the above key store.
- endpoint.link should be set to the URL of the MUI servers.

After editing this properties file, the program can be run by running the **MUISample.exe** executable generated by the build.

4.2.2.3 Anatomy of the Program

First, notice the **DemandResourcesMUIService.cs** file in the **MUISample** project. This code was generated by WCF from the WSDL and XSD files using the **SvcUtil.exe**, a tool included with Visual Studio. To generate this code, run the following command in the directory where the WSDL and XSDs are:

SvcUtil.exe /t:code /l:C# "*.wsdl" "*.xsd"

This command will generate the C# service stubs, as well as a configuration file that can be used to set up WCF. This example uses code to configure WCF, but the configuration file is another option. Additional documentation on the SvcUtil command can be found on MSDN at http://msdn.microsoft.com/en-us/library/aa347733.aspx.

The body of the program itself is defined in **Program.cs** in the **MUISample** project. In particular, the actual **DemandResourceMUI** instance is actually created in the main method:

```
        /*
         *  Address & Certifications for proxy...
         */

        EndpointAddress address =
            new EndpointAddress(properties["endpoint.link"]);

        X509Certificate2 clientKey = new X509Certificate2(
                // Client Side Certificate
                properties["client.key.file"],
                // file location
                properties["client.key.pswd"]
                // password
            );// end client certificate retrieval

        X509Certificate2 serverCertifications = new X509Certificate2(
                // server trusted certificates
                properties["srvr.trust.crt"]
                // file location
            );// end server trusted certificates retrieval

        /*
         *  Create proxy service
         */

        // proxy to use throughout
        DemandResourceMUI proxy = GetService(
            address,                    // Endpoint Address
            clientKey,                  // client key
            serverCertifications        // server trusted certifications
        );// end GetService
```

The **GetService** method then handles the details of instanciating a service proxy and hooking up the certificates:

```
/// <summary>
/// Creates the connecting proxy used by the remainder of the program.
/// </summary>
/// <param name="address">Endpoint Address for the proxy used.</param>
/// <param name="clientCert">Client Side Certificate</param>
/// <param name="serverTrust">Server Side trusted list of
                              certificates.</param>
/// <returns>Connecting Service</returns>
 private static DemandResourceMUI GetService(
        EndpointAddress address, X509Certificate2 clientCert,
                              X509Certificate2 serverTrust )
 {
        // Binding for proxy
        BasicHttpBinding binding = new BasicHttpBinding();

        // set security mode to Transport
        binding.Security.Mode = BasicHttpSecurityMode.Transport;

        // set credential type to require certificates
        binding.Security.Transport.ClientCredentialType =
            HttpClientCredentialType.Certificate;
```

```
        // create channel using binding and address
        ChannelFactory<DemandResourceMUI> factory
            = new ChannelFactory<DemandResourceMUI>(binding, address);

        // assign client side certificate
        factory.Credentials.ClientCertificate.Certificate = clientCert;

        // assign server trusted certificates
        factory.Credentials.ServiceCertificate.DefaultCertificate =
                    serverTrust;

        // create proxy connection to the server
        return factory.CreateChannel();
    }// end getService
```

4.3 Demand Resource Functions

The following section describes the format and usage for each of the demand resource functions in the MUI web service API.

4.3.1 Asset Interval Data

Query the baseline and performance data for a selected asset for all 5-minute intervals of the selected day.

4.3.1.1 Message Format

The query for asset interval data uses the following format:

```
<QueryAssetIntervalData day=”yyyy-mm-dd” assetId="xx" />
```

Element or Attribute	Data Type	Description
< QueryAssetIntervalData>	Complex	Root element of the query.
day	dateTime	Day that the asset interval report is for. This is a required parameter.
assetId	long	ID of a specific asset to query. If not specified, returns all assets controlled by the DDE, lead participant or meter reader.

4.3.1.2 Response Format

The system will respond with the following format:

```
<AssetIntervalDataResponse day="yyyy-mm-dd" dayType=”xxx”>
  <AssetIntervalData assetId="xxx">
    <AssetIntervalDataPoint
      time="?"
```

```
      unadjustedBaseline="?"
      baselineAdjustment="?"
      energyMarketLoadReduction="?"
      energyMarketNetSupply="?"
      capacityMarketLoadReduction="?"
      capacityMarketNetSupply="?"
      maxPotentialLoadReduction="?"
      maxPotentialNetSupply="?"
      curtailed="?"
      auditId="?"
    />
  </AssetIntervalData>
</AssetIntervalDataResponse>
```

Element or Attribute	Data Type	Description
<AssetIntervalDataResponse>	Complex	Root element of the report.
day	date	Day that the baselines report is for. The market day; specified in yyyy-mm-dd format using ISO8601 Date format
dayType	String	The day type of this report
<AssetIntervalData>	Complex	The interval data set for a particular asset.
assetId	long	The ID of the asset.
<AssetIntervalDataPoint>	Complex	The interval data value for a given interval.
time	dateTime	The time (5 minute interval) of the data value.
unadjustedBaseline	3-digit decimal	The most recent value of the unadjusted baseline for the asset for the time period.
baselineAdjustment	3-digit	The baseline

Element or Attribute	Data Type	Description
	decimal	adjustment MW.
energyMarketLoadReduction	3-digit decimal	The ISO-calculated load reduction used in the capacity market.
energyMarketNetSupply	3-digit decimal	The ISO-calculated net supply used in the capacity market.
capacityMarketLoadReduction	3-digit decimal	The ISO-calculated load reduction used in the capacity market.
capacityMarketNetSupply	3-digit decimal	The ISO-calculated net supply used in the capacity market.
maxPotentialLoadReduction	3-digit decimal	The ISO-calculated load reduction cap used in the reserve market.
maxPotentialNetSupply	3-digit decimal	The ISO-calculated net supply cap used in the reserve market.
curtailed	String	Value specifying if the asset was curtailed in this interval.
auditId	String	ISO-generated audit ID if the interval is part of an audit.

4.3.2 Asset Telemetry

DDEs and meter readers can use this message to query for the 5-minute asset telemetry values used to generate asset baselines.

4.3.2.1 Message Format

The query for asset telemetry has the following format:

```
<QueryTelemetry day="yyyy-mm-dd" assetId="xx"
                showOnlyBadQualities="true"/>
```

Element or Attribute	Data Type	Description
<QueryTelemetry>	Complex	Root element of the query
day	dateTime	Day to query for. If not specified, it defaults to today.
assetId	long	ID of a specific asset to query. If not specified, returns all assets controlled by the DDE or meter reader.
showOnlyBadQualities	boolean	Filter that can be used to return only the values with bad qualities.

4.3.2.2 Response Format

The system will respond with the following format:

```
<Telemetry day="yyyy-mm-dd">
   <AssetTelemetry assetId="xx">
     <TelemetryPoint time="yyyy-mm-ddTxx:00-05:00"
                     genMW="xx.xxx"
                     loadMW="xx.xxx"
                     genMWQuality="Good"
                     loadMWQuality="Good"
                     unadjustedBaseline="xx.xxx" />
   </AssetTelemetry>
    …
</Telemetry>
```

Element or Attribute	Data Type	Description
<Telemetry>	Complex	Root element of the report.
day	date	Day that the telemetry is for. Cannot be older than180 days in the past, otherwise an error message is returned; specified in yyyy-mm-dd format using ISO8601 Date format
<AssetTelemetry>	Complex	The telemetry for a specific asset. Will occur for each asset queried.

assetId	long	The ID of this asset
<TelemetryPoint>	Complex	Represents a telemetry point in time for this asset.
time	dateTime	The time (5-minute interval) of this point.
genMW	3-digit decimal	Generation MW
loadMW	3-digit decimal	Load MW
genMWQuality	Good, Bad	Quality value for the generation MW telemetry.
loadMWQuality	Good, Bad	Quality value for the load MW telemetry.
unadjustedBaseline	3-digit decimal	The unadjusted baseline in MW for this asset at this time.

4.3.3 Asset Telemetry Corrections

DDEs and meter readers use this message to submit corrections to 5-minute telemetry data for assets managed by the DDE or meter reader. DDEs and meter readers can correct these values for the past following the market rules.

4.3.3.1 Message Format

This message has the following format:

```
<SubmitTelemetryCorrections day="yyyy-mm-dd">
  <AssetTelemetryCorrection assetId="xx">
    <TelemetryCorrection time="yyyy-mm-ddTxx:00-05:00"
                         loadMW="+/-yy.yyy"
                         genMW="yy.yyy"
                         genMWQuality="Good"
                         loadMWQuality="Good" />
  </AssetTelemetryCorrection>
</SubmitTelemetryCorrections>
```

Element or Attribute	Data Type	Description
<SubmitTelemetryCorrections>	Complex	Root element of the submission.
day	dateTime	Market day to submit telemetry corrections for. For the sake of performance, only one day's worth of telemetry corrections can be submitted at once.

<AssetTelemetryCorrection>	Complex	A set of telemetry corrections for a given asset. May occur for each asset managed by the DDE.
assetId	long	ID for this asset
<TelemetryCorrection>	Complex	A telemetry correction for this asset.
time	dateTime	Time (5-minute interval) of this correction.
loadMW	3-digit decimal	The load MW for this interval. This is normally a negative number, but may be positive if the asset can pushback to the grid.
genMW	3-digit decimal	The generation MW for this interval.
genMWQuality	Good, Bad	Quality value for the generation MW telemetry.
loadMWQuality	Good, Bad	Quality value for the load MW telemetry.

4.3.3.2 Validation Rules

If a validation error occurs during the submittal, one or more of these error messages may appear within the <MUIFault> element returned in the error response:

Rule Name	Description	ErrorMessage
No telemetry for the future	Entries must be submitted before the current time.	Telemetry for time [time] is in the future and cannot be submitted
No duplicates	No duplicate time attributes can appear in any of the TelemetryCorrection elements.	Duplicate entry for time [time] and asset [id]
Assets must be owned by the user	The asset submitted must be managed by the user during the time specified in the Telemetry correction element.	Asset [id] is not owned by this user during [time]
Telemetry must be submitted in 5-minute intervals	The time attributes of all TelemetryCorrection elements must be on an even 5-minute boundary.	Telemetry for time [time] cannot be submitted because it is not an even 5-minute value

4.3.3.3 Query Format

The DDE will be able to query the submitted corrections for assets they manage using the following query message:

```
<QueryTelemetryCorrections day="yyyy-mm-dd" assetId="id" />
```

Element or Attribute	Data Type	Description
<QueryTelemetryCorrections>	Complex	Root query element
day	dateTime	The day to query corrections for. If not specified, defaults to today.
assetId	long	The ID of the asset to fetch for. This attribute is optional. If it is omitted, all assets managed by the DDE will be returned.

The response will consist of an element containing <AssetTelemetryCorrection> elements identical to the format described above:

```
<TelemetryCorrectionsQueryResponse day=”yyyy-mm-dd”>
  <TelemetryCorrection assetId="xx">
    …
  </TelemetryCorrection>
</TelemtryCorrectionsQueryResponse>
```

4.3.1 Resource Interval Data

Query the performance data for a selected resource for all 5-minute intervals of the selected day.

4.3.1.1 Message Format

The query for resource interval data uses the following format:

```
<QueryResourceIntervalData day=”yyyy-mm-dd” resourceId="xx" />
```

Element or Attribute	Data Type	Description
< QueryResourceIntervalData>	Complex	Root element of the query.
day	date	Day that the resource interval

		report is for. Specified in yyyy-mm-dd format using ISO8601 Date format
resourceId	long	ID of a specific resource to query. If not specified, returns all resources controlled by the DDE, lead participant or meter reader.

4.3.1.2 Response Format

The system will respond with the following format:

```
<ResourceIntervalDataResponse day="yyyy-mm-dd" dayType=”xxx”>
  <ResourceIntervalData resourceId="xxx">
    <ResourceIntervalDataPoint
      time="?"
      energyMarketLoadReduction="?"
      energyMarketNetSupply="?"
      capacityMarketLoadReduction="?"
      capacityMarketNetSupply="?"
      maxPotentialLoadReduction="?"
      maxPotentialNetSupply="?"
      auditId="?"
    />
  </ResourceIntervalData>
</ResourceIntervalDataResponse>
```

Element or Attribute	Data Type	Description
<ResourceIntervalDataResponse>	Complex	Root element of the report.
day	date	Day that the resource interval data report is for. Cannot be older than180 days in the past, otherwise an error message is returned; specified in yyyy-mm-dd format using ISO8601 Date format
dayType	String	The day type of this report
<ResourceIntervalData>	Complex	The interval data set for a particular

Element or Attribute	Data Type	Description
		resource.
resourceId	long	The ID of the resource.
<ResourceIntervalDataPoint>	Complex	The interval data value for a given resource.
time	dateTime	The time (5 minute interval) of the data value.
energyMarketLoadReduction	3-digit decimal	The ISO-calculated load reduction used in the capacity market.
energyMarketNetSupply	3-digit decimal	The ISO-calculated net supply used in the capacity market.
capacityMarketLoadReduction	3-digit decimal	The ISO-calculated load reduction used in the capacity market.
capacityMarketNetSupply	3-digit decimal	The ISO-calculated net supply used in the capacity market.
maxPotentialLoadReduction	3-digit decimal	The ISO-calculated load reduction cap used in the reserve market.
maxPotentialNetSupply	3-digit decimal	The ISO-calculated net supply cap used in the reserve market.
auditId	String	ISO-generated audit ID if the interval is part of an audit.