Electric Industry Data Exchange (EIDE)

Communications Protocol Document 1.0.5

Proposed by: WECC DEWG

November 22, 2006

Table of Contents

1.0 Introduction

1.1 Purpose

In 1998, the WECC DEWG members and various participants from the industry and vendor community worked together to begin the migration from the existing, aging, WECC X.25 communications network. The move was necessary due to the change in technology during the past decade, which marked the end of the production of the X.25 routing equipment used in the network. The WECC X.25 communications network is a synchronous 9.6 to 56 kbps modem and Motorola packet switch based private communications network. This network was established prior to the existence of the Internet that currently operates at speeds of 2 to 100 mbps. The packet switches have been out of production for several years and replacement parts are no longer available. The first phase of the migration was to move to Inter Control Center Communications Protocol (ICCP) to replace the "real-time" data exchange. The ICCP system was subsequently instigated and most entities that wished to exchange data at a high frequency installed ICCP communications. ICCP has subsequently been used to transmit EHV Data Pool data to the WECC, Security Coordinator data, Network Applications data, Reserve Sharing data, unit control data, and Operating Reserves Self Supply data. The migration from the WECC X.25 Process ID 8 (real time) data to ICCP block 1 and 2 (real time) data has been completed.

As of October 2001, replacement of the WECC X.25 Process IDs (PIDs) 1, 2,3,4,5,9,10, and 14 (as well as other PIDs) that were being used for periodic data exchange (meters, schedules, etc) had not yet been initiated. The Bonneville Power Administration, alarmed at the lack of spare parts for their existing systems, requested a meeting of the NWPP IDE-TC to discuss a migration plan. While block 8 of the ICCP communications system was designed to handle these types of objects, most entities were left with the responsibility of implementing block 8 on their own. The NWPP IDE-TC did not lead an effort to migrate to block 8. Now, new technology is available that has many advantages over block 8.

1.2 Scope

The NWPP IDE-TC designed the EIDE Communications Protocol in order to replace the remainder of the WECC X.25 communications protocol plus provide other functionality that seemed useful for BAs, PSEs, RAs, etc. Data with a periodicity greater than or equal to one minute may be exchanged with this protocol. Examples are schedule data, meter data, and other periodic power system data such as lake elevations, river flows, generator discharge, etc. The protocol also supports the exchange of messages and generic string data. EIDE can be easily extended privately for specific requirements between individual parties and the WECC DEWG may extend EIDE as requested by members.

1.3 Overview

EIDE is currently being used to exchange meter, power system, and schedule data between five entities with several more entities planning on implementation within the next few years. EIDE is an XML based communications protocol designed to conform to SMXP. SMXP (Simple Message eXchange Protocol) is a subset of SOAP (Simple Object Access Protocol) as defined by the W3C (World Wide Web Consortium). SMXP supports single-request-response message mapping using HTTP binding. The recommended implementation is to use TLS (Transport Layer Security) 128 (https) with client certificates and the Request-Response message mapping described in this document, however it is also possible to exchange the EIDE XML documents using other exchange mechanisms such as FTP, e-mail, etc. and emulate the synchronous single-request-response mapping using asynchronous response messages.

The SMXP functions include PUT and GET functions (a.k.a. Methods) for most data types, along with a server status function. The server status function can be used as a heartbeat or to request the functions that a particular server supports.

The EIDE schema supports a response code that can be used to provide error response information. Additionally the fault codes defined in the SMXP Style Guide (1.0) in section 3.4 may be used to provide the defined error responses to the host SMXP processor.

WECC DEWG recommends the use of SMXP as defined in the SMXP Style Guide[8] however the Message Exchange Model described in section 2 of the guide is extended to include other mechanisms of exchange as agreed to by the parties exchanging data. The EIDE Schema has been designed with this flexibility in mind. Any message exchange method that allows the exchange of ASCII data files is therefore suitable.

The message framework, encoding, XML conventions, etc. that apply to the EIDE communications protocol are described in the SMXP Style Guide version 1.0. All Date/Time data will be required to use the extended format in UTC (“2002-05-30T13:20:00.000Z”).

1.4 How SMXP Works

All EIDE messages are sent using the SMXP (Simple Method Exchange Protocol). This protocol is based upon a *remote procedure call* (RPC) paradigm. This means that instead of sending messages explicitly, you invoke procedures on remote machines, and pass any needed data as input parameters to the function. When the function is complete, it returns the result of its processing. The SMXP protocol is layered on top of the HTTP protocol, which handles all of the underlying communication. SMXP defines the set of rules for encoding remote procedure call parameters into HTTP POST messages, as well as the set of rules for how such messages should be processed by a remote server.
The steps of executing an SMXP method are as follows:

- A request is generated, containing the method name and any needed parameters.
- The request is sent via HTTP to a listener on the remote machine.
- The remote machine receives the SMXP request, examines it to determine which method or procedures should be executed, and validates the SMXP body (the XML payload) against a predefined schema (the EIDE schema in this case) to determine if it is SMXP and XML compliant.
- The remote machine then executes the appropriate method and packages the result into an SMXP compliant XML document.

Each SMXP method call has two important parts – the request and the response. Most of the methods used are *synchronous* methods, meaning that once the calling machine makes a request, it waits for a response containing the results of its request before continuing.

In numerous cases, *asynchronous* methods are used. In an asynchronous method, a request is generated and sent to a remote machine. The remote machine places the request into a queue, and sends a response to the calling machine that indicates the request has been received and queued for processing. The connection is then terminated. At some point in the future, the remote server runs the requested method and sends the result to the calling machine via a separate SMXP message (requiring a second request/response pair).

SMXP-compliant systems are only required to support the processing of one method call per connection session. Multiple calls per session are not supported.

SMXP is quickly evolving to be an electric industry standard for exchanging information over the Internet. It is, in fact, the model that E-tag V1.7 is built upon.

2.0 Schema Description

2.1 EIDE Schema Abstract Objects

The EIDE data exchange schema set is composed of abstract object types, each of which may be composed of any combination of elements, attributes, and complex types. While the objects were created specifically for exchanging certain types of data, this is not meant to be an imposed limit.

The EIDE data exchange schema abstract object types are:

ScheduleType	For exchanging schedule data
MeterType	For meter data
PowerSystemDataType	For power system data (such as lake levels, temperatures, etc.)
MessageType	For sending messages
StringType	For sending generic ASCII data

PIDType	For sending old X.25 PID 1,2,3,4,5 and 14 data

These objects are then embedded within SMXP function definitions.

The schema also defines several other types and referenced elements that are used to abstract objects used within the data exchange elements.

2.2 EIDE Schema SMXP Functions

The protocol allows for both synchronous and asynchronous message responses. Each type therefore has two sets of single-request-response messages associated with both the Put and Get functions. For example, if an entity initiates a PutSchedule to a partner entity, the partner would respond with a PutScheduleResponse. If the sending entity requests an acknowledgement, then the partner entity would initiate a PutScheduleAck indicating whether the schedules had been processed correctly or not and would expect a PutScheduleAckResponse in response to the PutScheduleAck. Documents are sent to the remote partner using the http(s) POST method.

The remote procedure call functions are listed below by object type:

ScheduleType:

- PutSchedule
- PutScheduleResponse
- PutScheduleAck
- PutScheduleAckResponse

- GetSchedule
- GetScheduleResponse
- GetScheduleAsyncReply
- GetScheduleAsyncReplyResponse

MeterType:

- PutMeter
- PutMeterResponse
- PutMeterAck
- PutMeterAckResponse

- GetMeter
- GetMeterResponse
- GetMeterAsyncReply
- GetMeterAsyncReplyResponse

PowerSystemDataType:

- PutPSD
- PutPSDResponse

PutPSDAck
PutPSDAckResponse

GetPSD
GetPSDResponse
GetPSDAsyncReply
GetPSDAsyncReplyResponse

MessageType:

PutMessage
PutMessageResponse
PutMessageAck
PutMessageAckResponse

There is no “get” function for messages.

StringType:

PutString
PutStringResponse
PutStringAck
PutStringAckResponse

GetString
GetStringResponse
GetStringAsyncReply
GetStringAsyncReplyResponse

PIDType:

PutPID
PutPIDResponse
PutPIDAck
PutPIDAckResponse

GetPID
GetPIDResponse
GetPIDAsyncReply
GetPIDAsyncReplyResponse

This list is representative of the methods implemented by EIDE however the schema itself is the definition of correctly formatted EIDE XML. The schema contains additional objects that are not listed above and will change from time to time as approved by the WECC DEWG.

2.3 EIDE Example Schema for PutSchedule Object

The schema is shown below in text format. The easiest way to view and understand the schema is graphically as it shows the hierarchical layout of the PutSchedule object. The fully expanded graphic does not fit on one screen; therefore, a partial graphic is shown in section 2.3.5. The schema is best viewed using a schema-viewing tool such as XML Spy[9].

2.3.1 Put Schedule Element

```
   <xs:element name="PutSchedule">
      <xs:annotation>
         <xs:documentation>Send schedule(s)</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="MessageInfo" type="MessageInfoType"/>
            <xs:element name="Schedules" type="ScheduleType"/>
         </xs:sequence>
      </xs:complexType>
    </xs:element>
```

2.3.2 MessageInfoType

```
<xs:complexType name="MessageInfoType">
      <xs:annotation>
         <xs:documentation>Message Header</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="SysGenID" type="xs:int">
            <xs:annotation>
               <xs:documentation>Unique, increasing integer number or, in sync response, ID in
request</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="TimeStamp" type="xs:dateTime">
            <xs:annotation>
               <xs:documentation>Date/Time in UTC that this document was created</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="Sender" type="xs:string">
            <xs:annotation>
               <xs:documentation>NERC ID of the Sending Party</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="Receiver" type="xs:string">
            <xs:annotation>
               <xs:documentation>NERC ID of the Receiving Party</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="EntityCode" type="xs:string">
            <xs:annotation>
               <xs:documentation>NERC ID of the Party on whose behalf the Sender is
acting</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="ProcessID" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>Optional field cooresponding to the WECC PID</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="DataSet" type="xs:int" minOccurs="0">
```

```
                <xs:annotation>
                    <xs:documentation>Optional field cooresponding to an integer identifying the data
set</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="ListID" type="xs:int" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Optional field cooresponding to an integer identifying a particular
list</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="ResponseSysGenID" type="xs:int" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>SysGenID that this message is responding to</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="Comment" type="xs:string" minOccurs="0"/>
            <xs:element name="RequireAck" type="xs:boolean" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Set to YES if response "ProcessedOK" is required.  Ignored if this is a
response.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="UserID" type="xs:string" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Name or ID of the sender</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
```

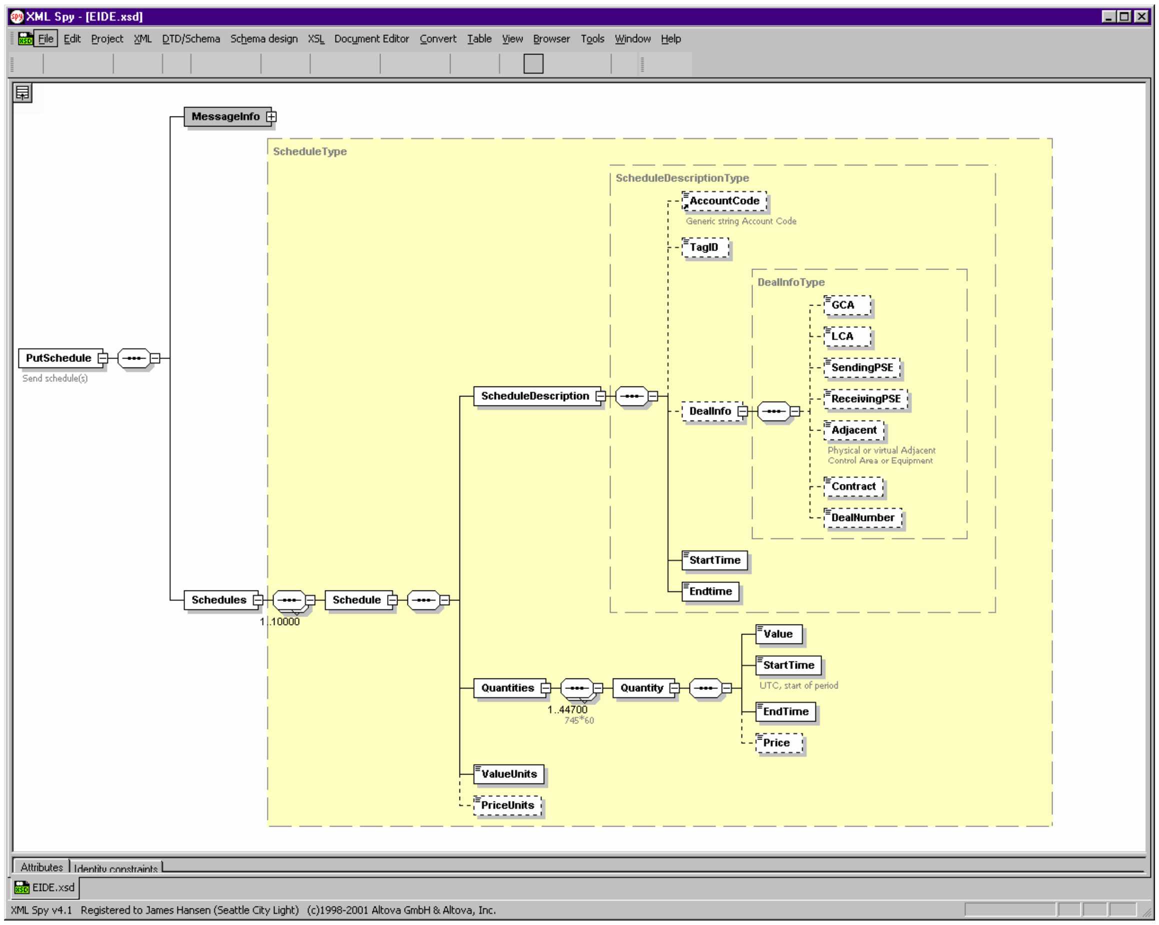
2.3.3 ScheduleType

```
<xs:complexType name="ScheduleType">
        <xs:annotation>
            <xs:documentation>Schedules etc.</xs:documentation>
        </xs:annotation>
        <xs:sequence maxOccurs="10000">
            <xs:element name="Schedule">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ScheduleDescription" type="ScheduleDescriptionType"/>
                        <xs:element name="Quantities">
                            <xs:complexType>
                                <xs:sequence maxOccurs="44700">
                                    <xs:annotation>
                                        <xs:documentation>745*60</xs:documentation>
                                    </xs:annotation>
                                    <xs:element name="Quantity">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="Value" type="xs:int"/>
                                                <xs:element name="StartTime" type="xs:dateTime">
                                                    <xs:annotation>
                                                        <xs:documentation>UTC, start of
period</xs:documentation>
                                                    </xs:annotation>
                                                </xs:element>
                                                <xs:element name="EndTime" type="xs:dateTime"/>
                                                <xs:element name="Price" type="xs:float" minOccurs="0"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="ValueUnits">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="MW"/>
                                    <xs:enumeration value="MWh"/>
                                    <xs:enumeration value="kWh"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="PriceUnits" minOccurs="0">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="USD"/>
                                    <xs:enumeration value="CDN"/>
                                    <xs:enumeration value="Pesos"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
```

2.3.4 DealInfoType

```
    <xs:complexType name="DealInfoType">
        <xs:annotation>
            <xs:documentation>Schedule Information, at least one of these fields must be
resent</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="GCA" type="xs:string" minOccurs="0"/>
            <xs:element name="LCA" type="xs:string" minOccurs="0"/>
            <xs:element name="SendingPSE" type="xs:string" minOccurs="0"/>
            <xs:element name="ReceivingPSE" type="xs:string" minOccurs="0"/>
            <xs:element name="Adjacent" type="xs:string" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Physical or virtual Adjacent Control Area or
Equipment</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="Contract" type="xs:string" minOccurs="0"/>
            <xs:element name="DealNumber" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
```

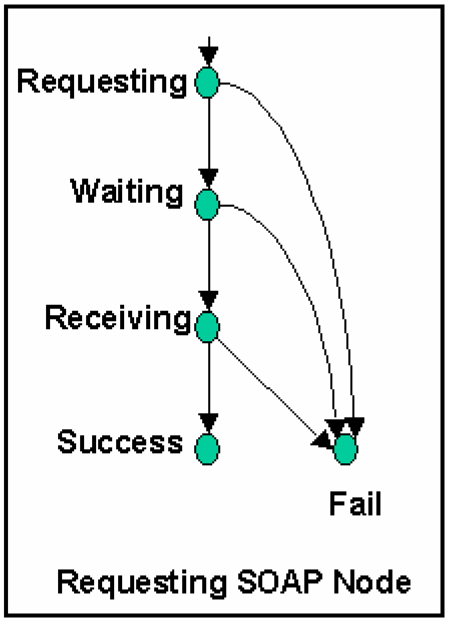
2.3.5 Partial Graphic of PutSchedule Object in XML Spy

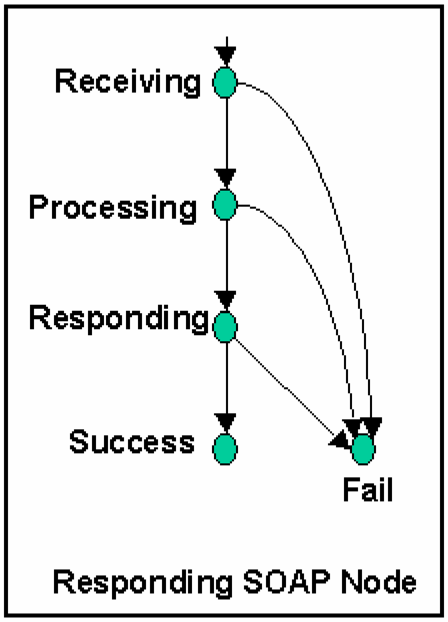


3.0 Data Exchange Message Mapping

The HTTP binding is described in the discussion below using the single-request-response message mapping described in the SOAP W3C documentation[2,3,4]. Other transport methods are not discussed, however the same general principles apply. In the SMXP HTTP binding message mapping, the SMXP methods defined by the EIDE schema are passed as the body to an SMXP HTML document.

3.1 Single-Request-Response State Diagram





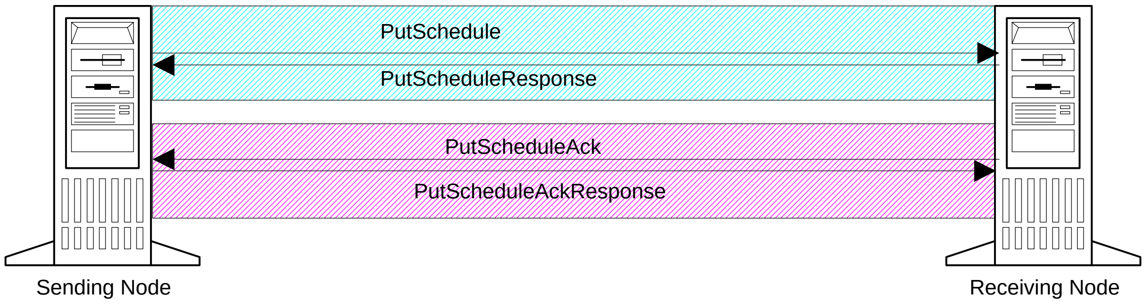
(From SOAP Version 1.2 Part 2: Adjuncts section 7.1.3)

The single-request-response message mapping state diagram is shown above. The "Requesting" SOAP node issues an HTTPGet using POST (basically what a web browser does when you enter a URL with parameters and hit return) and the "Receiving" SOAP node replies to the HTTPGet with an HTTPResponse. The HTTPGet function generally waits until the receiving node responds and returns the response object as part of the return value of the get function. The HTTPGet function will take at least the target URL and SMXP method (XML document) as an argument. The receiving node should reply with an SMXP method.

See the SOAP Adjuncts document[4] section 7.1.3 for the full state transition tables of sending and receiving node.

3.2 Asynchronous Communications Example of PutSchedule

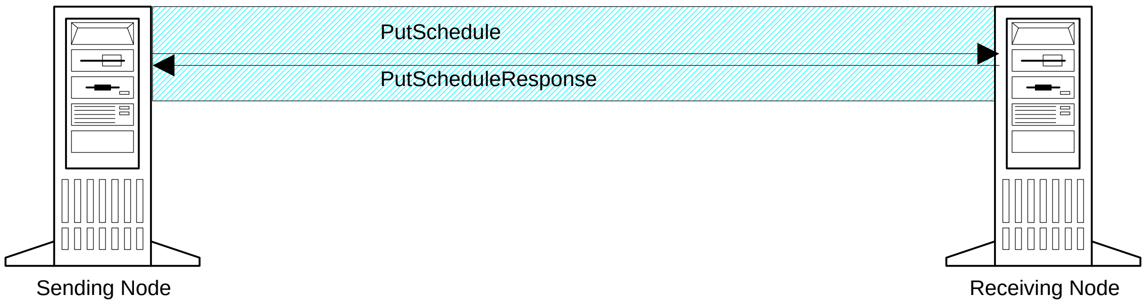
The sending node puts schedules to the receiving node and RequireAck field is set to "YES". This exchange is accomplished using the PutSchedule method, which is responded to with the PutScheduleResponse method.



The receiving node responds initially with the ResponseCode set to "QueuedOK" in the PutScheduleResponse XML document. The receiving node then processes the schedules, most likely saving them to a database or inserting them into a scheduling system, and then issues an HTTPGet function passing a PutScheduleAck in the body of the SMXP XML document. The response SysGenID field is populated with the SysGenID of the original PutSchedule and the ResponseCode is set to "ProcessedOK". The receiving party responds synchronously with a PutScheduleAckResponse.

3.3 Synchronous Communications Example of PutSchedule

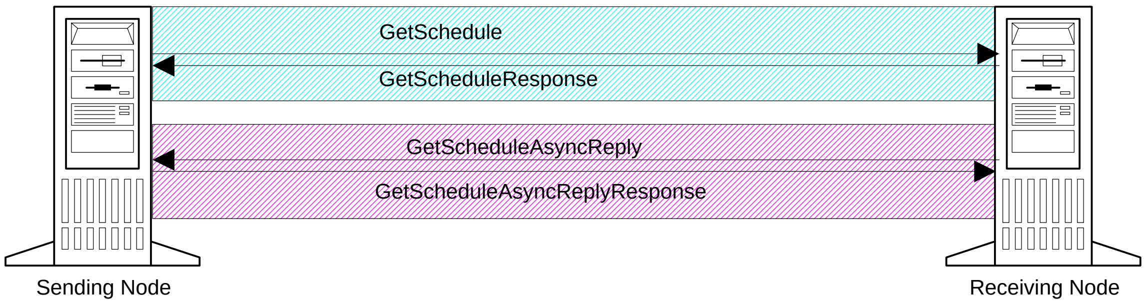
The sending node puts schedules to the receiving node and the RequireAck field is set to "YES". In this example, the receiver is capable of immediately fully processing the schedules and responds with "ProcessedOK".



Note, this example would look the same if the sender either set the RequireAck field to "FALSE" or didn't set the RequireAck field. In that case, the sender would only care about "ReceivedOK" being set in the ResponseCode.

3.4 Asynchronous Communications Example of GetSchedule

The sending node uses HTTPGet with a GetSchedule SMXP method. By convention, the message header contains a DataSet, ListID or neither set. If neither are set, then all schedules that can be requested by the requestor are sent in the response. If either is set, then the schedules corresponding to the ListID or DataSet are returned. The first ScheduleDescription field populated with the StartTime and EndTime so that the receiver can determine the date range the requested is requesting. The GetScheduleResponse is then sent back by the receiving node with a "ReceivedOK" response.



The receiving node then gathers up the requested schedules and becomes the sending node in the exchange. The receiving node issues an HTTPGet with a populated GetScheduleAsyncReply XML element in the body element of an SMXP XML document. The receiver replies with a GetScheduleAsyncReplyResponse.

3.5 Synchronous Communications Description of GetSchedule

The sending node uses HTTPGet with a GetSchedule SMXP method. The GetScheduleResponse is then sent back by the receiving node with all schedule data requested by the requesting party populated.

4.0 EIDE SMXP Data Communications Protocol Conventions

Certain conventions must be followed in order for the sending and receiving nodes to be able to communicate. This section describes those conventions.

4.1 SysGenID Conventions

The SysGenID is defined as an integer that limits the minimum and maximum range from -2147483648 to 2147483647. Each MessageInfo header contains a SysGenID and by convention, this SysGenID is required to be unique by entity and increasing (not necessarily contiguous). The SysGenID is assigned to every EIDE XML document and can be used to uniquely identify the document when used in combination with the Sender. Note that while it will probably take quite a while to reach the maximum int size, that TimeStamp must be used to uniquely identify the documents unless they are aged into a history table. The complete unique identification of a document will then consist of Sender, SysGenID, and TimeStamp. When the SysGenID reaches the 2147483647 maximum it should cycle to the original starting point, which could be either negative or positive. The DEWG recommends starting at zero and not using the negative range except for special purposes such as internal message exchange or testing.

In a Request/Response message pair, the requesting node will generate a unique SysGenID to populate this field and the responding node will use the requestor's SysGenID to populate the ResponseSysGenID field in the response. The ResponseSysGenID field will also be set in an Ack or AsynchReply message.

4.2 ResponseSysGenID Conventions

This optional field is required in all Ack and AsyncResponse messages to allow the receiving node in the exchange to uniquely identify the message to which this response corresponds. It is required to be the SysGenID of the message that the AsyncResponse is in response to. It is not required in the response method to an HTTPGet, therefore it is not a required field in any header. It is up to the party receiving the Ack or AsyncResponse to validate this field, match it with the expected response, and enforce the requirement for receipt since it cannot be required in the XML.

4.3 Identifying Data Conventions

In order for the receiving party to identify the data that is being sent to it or requested from it, the sending party must fill in the DataSet, ListID, AccountCode, or DealInfo. All of these fields are optional in the schemas. At least one must be populated in order for the parties to identify the data. Most parties will use look up tables to map the DataSet, ListID, or AccountCode to local definitions of the data. It may be possible for parties to directly map the data if the DealInfo element is populated.

If DataSet or ListID are used as the sole identifier for the data, then the data must be presented in the specific order agreed to by the parties so that sequences occurring multiple times can be mapped to specific local definitions. WECC account code mapping is generally preferable.

4.4 Acknowledgement Conventions

Every HTTPGet must receive some kind of ResponseCode in the HTTPReply in order to be considered successful. The DEWG recommends that the sending party, at a minimum, look for the “ReceivedOK” response and take corrective action if this response is not received. Complete definition of corrective action is outside the scope of this document however, examples are; (1) waiting for some period, then attempting to retransmit the data; and/or (2) notifying the local staff that there has been a transmission problem.

If the sender requests an acknowledgement by setting the RequireAck field to “YES”, then the receiver should reply with “ProcessedOK” after the EIDE XML document has been processed successfully by the receiving node. Note that setting RequireAck to “YES” in any response SMXP method is not defined as part of the EIDE protocol and the receiver may choose to ignore the RequireAck field in all response methods.

4.5 Get Methods Conventions

In order to identify the data that is being requested, and the time period for which it is being requested, certain conventions need to be followed to implement the "Get" methods. The EIDE protocol was designed in order to provide the maximum amount of flexibility for parties to exchange data. This actually creates some problems in one sense because the burden falls to the implementer to write appropriate algorithms to accommodate this flexibility. For get functions, the protocol was designed to allow the requestor to ask for all applicable data, data identified by a list, data identified by a dataset, data identified by account code, or data identified by DealInfo information. Implementation of all these options is not necessary between parties that agree to specific conventions however. The NWPP parties have agreed to populate the ListID to specify a particular list, the DataSet to identify a particular DataSet, or leave both out of the document to mean "all applicable data". The NWPP parties also agreed to populate the first "description" node (schedule description, etc) with the start and end dates of the data request.

4.6 Put Meter Conventions

Parties in the NW have implemented the PutMeter method to transmit data for the previous hour ending with the CumTotal field containing the total for data for that account between 01:00 and the last hour ending. For example, if the PutMeter is sent at 08:01 then the quantity field contains data for HE 8 and the CumTotal field contains the sum of quantities for HE 1 to HE 8. Generally, the Put Meter method is used to send only one hour of data at a time but may be used to send multiple hours of data. PutMeter methods for previous days will usually contain 24 hours of data.

4.7 Put Schedule Conventions

Parties in the NW have implemented the PutSchedule method to transmit hourly data for the full day, HE 1 to HE 24.

4.8 StartTime and StopTime Conventions

Start Time is the hour beginning time and stop time is the hour ending time. A full days schedule for today, in local time, would have a start time of today 00:00 and an end time of tomorrow 00:00. Similarly the start and stop time for hour ending 08:00 data would be 07:00 and 08:00 respectively.

4.9 Data Compression Conventions

When multiple quantities are sent for a single account, they can be compressed if the quantities are the same. Such compressed profiles reduce message transfer sizes. For example, a schedule that contains 0s for HE 1 to 5, 10s for HE 6 to 8, 20s for HE 9 to 24 would be sent with 3 quantity blocks. One with a start time of 00:00 and end time of 05:00 with quantity 0, one with start time of 05:00 and end time of 08:00 with quantity 10, and one with start time of 08:00 and end time of 00:00 (the next day) with quantity 20.

5.0 Security, Identity, and Authenticity

Security (no one else can read the data except for the intended recipient), identity (the parties involved in the communication are who they claim to be), and authenticity (the data received is identical to the data sent) all can be achieved using TLS and requiring client certificates. The "NWPP DETC Periodic Data Exchange" abstract[10] addresses security issues and can be referenced for further information. The DEWG recommends TLS 128 and requires the use of NAESB PKI compliant certificates. NAESB PKI compliant client certificates are included in secure TLS POST calls to Internet servers implementing TLS using NAESB PKI compliant certificates. The DEWG recommends mapping the client certificates using common name, etc. to eliminate the maintenance burden of exchanging specific certificates. The DEWG also recommends building the EIDE software so that it automatically switches between secure and plain text (https/http) transfers depending on the URL of the target.

TLS encryption requires simply that the software initiating the connection use an TLS call. Web clients (like Internet Explorer) use TLS whenever they see "https" in the URL. Internally, IE is using a different method to invoke the session. The TLS method that gets invoked establishes communication with the remote server first and a session key is exchanged which allows the client to encrypt data and the target server to decrypt the data. In a phased approach, server side TLS can be completed first where no client certificate is required. This accomplishes the goals of 1) ensuring that no one can read the data except for the intended parties and 2) ensuring the data has not been manipulated in transit. This also ensures server side authentication.

TLS authentication on the client side is accomplished by the attachment of a client certificate to the message. Passing a parameter to the session establishment call containing the fully qualified file name and the certificate password does this. There are several different methods that can be used by the web server to authenticate the client cert, including matching the cert. The recommended method is to map based on common name and certificate level.

NAESB PKI standards require the use of certificates that meet specific criteria contained in the standard. Generally, OATI certificates, CAISO certificates, and Verisign level 3 certificates meet the certificate producer standards. The standards also require the certificate holders to adhere to specific requirements. The basic intent of these standards is that they are kept secure and cancelled as soon as possible if compromised. This includes employee termination and security breach events.

The use of CAISO or OATI certificates will require special installation on the web server that needs to accept or use those certificates. Trusted Root Certification Authorities and Intermediate Certification Authorities need to be updated to include these cert chains. There are various methods to accomplish this based on the web server that is being used.

The client certificate is generally installed on a web browser (client) and then exported to a file. When creating the client cert file, include the private key, export everything in the path, and remember the password, as this will be used in the method call used to establish the session. Ensure that the file is installed on the server issuing the method call and has permissions that allow it to be accessed by the software using the method.

6.0 Design Considerations

The authors have found that the interfaces between the EIDE communications system and the back office (EMS system, scheduling system, etc) systems should be designed with the a few key ideas in mind.

- Create a data buffer mechanism to handle the case where data has been received by the EIDE interface but the back office system is unavailable. This provides for a graceful recovery in the case where the system is taken offline for a short time or the network connection is down.

- Assume that your outbound network connection may hang if there is a problem at a receiving parties site and build in a timeout routine of some sort so that you application will continue to function. The timeout period should be 60 seconds.

- Do not send out your Put methods if your back office systems are down and unable to provide the data. Consider generating an error message that is displayed to the back end system user so that they may take appropriate action.

- Design the system to update data or provide data only within specific timelines as specified by back end system users or business rules.

- To facilitate debugging efforts, the outgoing XML messages should be formatted into a form that is easily readable by the receiving parties. The messages should not be chunked into long lines of information but should instead be “pretty”. See Section 2.3 above for examples of what the messages should look like at the receiving end(s).

- “Null” responses and/or “html” responses should be considered to be failures. Only valid XML traffic should be considered as a valid response.

7.0 References

1	W3C Document Home Page	http://www.w3.org/TR/
2	SOAP Primer	http://www.w3.org/TR/soap12-part0/
3	SOAP Messaging Framework	http://www.w3.org/TR/soap12-part1/
4	SOAP Adjuncts	http://www.w3.org/TR/soap12-part2/
5	XML Primer	http://www.w3.org/TR/xmlschema-0/
6	XML Structures	http://www.w3.org/TR/xmlschema-1/
7	XML Data Types	http://www.w3.org/TR/xmlschema-2/
8	SXMP	http://server06.nerc.com/tag/E-tag/smxpv1-20010408.doc
9	XML Spy	http://www.xmlspy.com/
10	NWPP IDE-TC Abstract	DETC Periodic Data Exchange December, 2001

8.0 Sample Documents

Sample XML documents with header information and SOAP wrappers are provided separately as text documents.

Appendix 1 — Implementation Guidelines and Practices

The document below has been developed after several years of EIDE data-link experience and is included as an aid in implementing the EIDE protocol.

EIDE Implementation Guidelines and Practices

The Electric Industry Data Exchange (EIDE) protocol is the WECC standard data exchange protocol for exchanging non-realtime periodic data such as meter data, schedule data, and other power system data. The protocol standardizes the method by which all WECC members may exchange non-realtime data. EIDE will be used by the WECC for the WECC Interchange Tool (WIT). Many Balancing Authorities in the WECC will be implementing an EIDE-compliant data link to upload to and/or download from the WIT both their scheduled and actual interchange values within the next year.

The EIDE protocol has been used to exchange meter, power system, and schedule data between several entities in the Pacific Northwest for the past three years.

The XML schema is a powerful and flexible definition tool that allows solutions to be developed relatively quickly. This flexibility has some ambiguities that can be (and are) applied in ways that are inconsistent. In other words, just because the EIDE XML schema allows a particular message stream or structure, it is not always supported by all entities implementing EIDE.

This document attempts point out some of the implementation issues that entities implementing EIDE have encountered over the years so that others are aware of them. It also attempts to develop and define some business guidelines and/or procedures that have worked well over the past three years. These business guidelines are suggestions only and in no way should restrict anything that parties may agree upon. It is hoped that lessons we've learned thus far, and discussed here, will help any new EIDE data link installations.

Data Methods

EIDE supports numerous message exchange structures. These structures are officially called "Methods" because they define the method or remote procedure that is to be implemented on the remote system to handle that particular inbound message structure.

For the hourly exchange of metered and scheduled interchange information, there are four methods that will be commonly utilized; the PutSchedule, PutMeter, GetSchedule, and GetMeter methods. One would think that the PutSchedule method is only used for scheduled interchanges and that the PutMeter method is only used for metered interchanges. However, that is not a requirement of the protocol:

- PutSchedule—The PutSchedule method may be used to exchange any hourly values, whether they are schedules, actuals, nets, memo accounts, generation, loads, etc.

    - ✓ **Guideline 1—Allow the use of the PutSchedule method for any type of hourly accounts. Do not restrict it to "schedules" only.**

  The EIDE XML schema does allow for any number of hours to be sent with the PutSchedule method—even a single hour at a time. In other words, even though it is designed to handle multiple hourly accounts for multiple hours (single hour to multiple days' worth of data) it is generally implemented so that a single day's worth of hourly values is transmitted. Backend systems are also most likely set up to only expect and handle whole days (24 hours) at a time.

- ✓ **Guideline 2—Send whole days at a time with the PutSchedule method.**

- ✓ **Guideline 3—Whole days should start and end on day boundaries—even on time change days.**

- PutMeter—The PutMeter method may be used to exchange any type of hourly values, whether they are schedules, actuals, nets, memo accounts, generation, loads, etc.

  - ✓ **Guideline 4—Allow the use of the PutMeter method for any hourly accounts. Do not restrict it to "meters" only.**

  Again, the EIDE XML schema does allow for any number of hours to be sent with the PutMeter method—even a whole day at a time. Even though it could handle multiple hourly accounts for multiple hours, it is generally used to transfer only a single hour at a time. Backend systems are also most likely set up to only expect and handle a single hour at a time.

  - ✓ **Guideline 5—Send one hour at a time with the PutMeter method.**

- GetSchedule—the GetSchedule method allows a user to query a remote system for data for multiple accounts for multiple hours. The schema allows for any number of hours to be requested. Since the data is to be returned in the equivalent of a PutSchedule method, only whole days should be requested. It can also be used to request any type of hourly accounts.

  - ✓ **Guideline 6—Request whole days at a time with the GetSchedule method.**

  - ✓ **Guideline 7—The GetSchedule method can query for any type of hourly accounts (schedules, meters, generation, etc.)**

- GetMeter—the GetMeter method allows a user to query a remote system for data for multiple accounts for multiple hours. Generally only a single hour at a time is requested. However, it certainly may be utilized to request a full day's worth of data.

  - ✓ **Guideline 8—Generally, request a single hour at a time with the GetMeter method.**

  - ✓ **Guideline 9—The GetMeter method can query for any type of hourly accounts (schedules, meters, generation, etc.)**

Data Messages

The EIDE XML schema allows for multiple hourly accounts to be sent within a single transfer message stream. There is no reason they should to be transferred individually. Yet some companies have limited this capability, for whatever reason, by setting restrictions such that there can only be one hourly account per transfer message. This of course is inefficient and difficult to handle cleanly.

- ✓ **Guideline 10—As many hourly accounts as possible should be transferred in a single transfer message stream.**

Schema Validation Issues

One of the great features of using XML is that the message stream may be validated at any time—at the sending end and/or the receiving end—in order to check that it is "well-formed" and that all of its data values are included and of the correct type. If the message fails to comply with all schema validation rules and formats, the transfer is aborted and/or rejected.

All EIDE traffic is most certainly validated on the receiving end of a transfer. However that does not seem to be the case on the sending end of a transfer. There have been numerous occasions when traffic arrives that is rejected because of a schema validation issue. (For example, optional fields are sent in the message that has no associated data for the field, i.e., “null” data). These schema validation errors should be caught before the message is ever sent out.

- ✓ **Guideline 11—All outgoing EIDE traffic should be validated against the EIDE XML schema before being transmitted.**

Generally, EIDE traffic is sent from a remote server and so the rejection error messages returned by the receiving end may never seen by the users themselves.

- ✓ **Guideline 12—Alarms and/or logs should be set up to alert users or IT staff of any validation or transfer problems.**

Additionally, as EIDE traffic is received, a message acknowledgement is sent back to the sender that either reports that the message was received successfully or was rejected for what ever reason. Depending on implementation, users generally have no way of knowing or viewing this type of information.

- ✓ **Guideline 13—Log viewing facilities should be made available to users to be view traffic acknowledgements and link status reports. This would significantly assist users and reduce downtime.**

Schedule Description

Each EIDE PutSchedule method (as do others) has a complex data element called a “Schedule” which is used to define the parameters of a “schedule”. This element contains:

- A reference number (account code, Tag ID, etc.)
- The StartTime and EndTime of the schedule;
  - Note: these are required fields.
  - Schedule should start and stop on whole day boundaries.
- The Quantity data element (aka, profile) of the schedule.
  - A Value (i.e., the MWH flowing for this period)
  - The StartTime and EndTime for the given Value.

There have many instances when the schedule’s profile is inconsistent with the schedule’s defined start and end times. For example,

- A profile actually falls outside the timeframe defined by the StartTime and EndTime of the Schedule.
- The first profile does not start on the StartTime of the Schedule.
- The last profile does not end on the EndTime of the Schedule.

None of these problems are trapped out with schema validation. But these inconsistencies cause all sorts of ambiguities and misinterpretations of the data such that wrong values end up in wrong hours causing all sorts of problems. Hence:

- ✓ **Guideline 15—The StartTime of the first defined profile element (Quantity) and the EndTime of the last defined profile element should match the StartTime and EndTime of the Schedule.**

- ✓ **Guideline 16—All hours between the StartTime and the EndTime of the Schedule should be included and defined by the appropriate profile elements.**

- ✓ **Guideline 17 – Use compression whenever possible. This is where StartTime and EndTime will define a range using the same values for each Quantity.**

Other Guidelines

The SOAP Action field has many possible formats. For the EIDE protocol, use the ":" notation (example: EIDE:PutSchedule). This is not the Microsoft default and so may require method modification.

- ✓ **Guideline 18 – Use the ":" notation for Soap Action. "EIDE:<MethodName>". Example: EIDE:PutSchedule.**

UTC Time may be represented either in local time with an offset or in Zulu time. For the EIDE protocol, use the Zulu representation with Time ending in "Z". Example: <TimeStamp>2006-10-31T23:17:31Z</TimeStamp>.

- ✓ **Guideline 19 – Use the UTC Zulu time representation.**

Conclusion

The guidelines and rules discussed above are not specifically directed at the WIT implementation requirements. In fact, WIT implementers might have something else entirely in mind. However, as entities implement EIDE for their WIT requirements, they will also realize that EIDE is valuable tool for hourly data transfers between ALL entities—Balancing Authority neighbors, generation partners, reliability centers, etc. It only makes sense that all EIDE transfers are as consistent as possible, regardless of the application. By being aware of the issues we have experienced in the past and adhering to these guidelines, it may help make all implementations go more smoothly and trouble free. Again, these are only guidelines, not requirements.