

PJM RPM

External Interface Specification

Date: June 1st, 2007

Revision 1

Transmittal No.:

AREVA
Redmond, Washington USA

Copyright © 2002, 2006 AREVA. All Rights Reserved.

ESCA, HABITAT, and PC17 are registered trademarks.

EMP, ESCAHelp, ESCATOOLS, Genesys, HABConnect, NETIO, OpenAccess Gateway, PC Rapport, ProGraf-RT, QuickConvert, QuickMap, Rapport-FG, RDR, and TrakR are trademarks of ALSTOM ESCA Corporation.

Exceed, Exceed MDK, and Exceed XDK are trademarks of Hummingbird Communications Ltd.

HyperHelp is a trademark of Bristol Technology, Inc.

IBM, OS/2, and PC are registered trademarks and PCXT is a trademark of International Business Machines Corporation.

Microsoft, Windows NT, Windows 98, MS-DOS, and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation.

MicroVax, ULTRIX, VAX, VMS, and VT are registered trademarks and Alpha AXP, CDA, DEC, DECnet, DEComni, DECpc, DECstation, DECwindows, Compaq Tru64 UNIX, OpenVMS, Rdb/VMS, VAX DEC/CMS, VAXstation, and VAX/VMS are trademarks of Compaq Computer Corporation (Compaq).

Monotype and Monotype Century Schoolbook are trademarks of Monotype Typography Ltd. Book Antiqua, Century Gothic, and Bookman Old Style are trademarks of The Monotype Corporation.

ObjectStore is a trademark of Object Design, Inc.

PSE Pro is a trademark of eXcelon Corporation.

ORACLE is a registered trademark of Oracle Corporation.

Orbix and OrbixTalk are registered trademarks of Iona Technologies PLC.

OSF/1 and OSF/Motif are registered trademarks of Open Software Foundation.

Postscript is a registered trademark of Adobe Systems, Inc.

Razor is a registered trademark of Tower Concepts, Inc.

Rogue Wave and Tools.h++ are registered trademarks of Rogue Wave Software, Inc.

True DBGrid is a trademark of Apex Software Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

VideoSoft is a registered trademark of VideoSoft.

X Window System is a registered trademark of the Massachusetts Institute of Technology.

xSTRA STREAMS is a trademark of UconX Corporation.

All other registered or trademarked names and products are property of their respective owners.

Revision History

Revision	Transmittal #	Date	Comments
1		June 1, 2007	[dbs] Original Preliminary Draft.

Table of Contents

PJM RPM	III
1 INTRODUCTION	1
1.1 Purpose.....	1
1.2 Technology Prerequisite Knowledge	1
1.3 Document References	1
1.4 Terminology.....	2
2 INTRODUCTION TO THE PROGRAMMATIC API	6
2.1 Messaging Overview.....	6
2.1.1 The Role of the Participant	6
2.1.2 The Role of PJM.....	6
2.2 Request/Reply Messaging	7
2.3 Web Technologies.....	8
2.3.1 XML – Extensible Markup Language.....	8
2.3.2 SOAP – A Web Services API.....	10
2.3.3 HTTP/HTTPS	11
2.3.4 SSL and Authentication	11
3 COMMON PRINCIPLES	12
3.1 RPM Participant Registration.....	12
3.2 RPM Server Addressing.....	12
3.3 HTTP Command and Headers.....	12
3.3.1 HTTP Request Message Format	13
3.3.2 HTTP Response (reply) Message Format	13
3.4 SOAP Envelope	14
3.5 XML Namespaces	15
3.6 XML Schema.....	15
3.7 Error Response	16
3.8 XML Data Type and Specification Semantics.....	17
3.8.1 Character Case.....	17
3.8.2 XML Boolean Data.....	17
3.8.3 XML Date and Time.....	18
3.8.4 XML Character Data and Markup and Escaped Characters.....	19
3.8.5 XML Null Data.....	20
3.8.6 XML Absent Elements	20
3.9 File Upload and Download Format	21
4 SUBMIT REQUEST	23
4.1 Submitting a Resource Offer into an Auction	24
4.1.1 Purpose	24
4.1.2 Message Format.....	24
4.1.3 Response Message	26
4.2 Submitting Load Contribution Data	27
4.2.1 Purpose	27
4.2.2 Message Format.....	27
4.2.3 Response Message	28
5 ERROR RESPONSE	30

1 Introduction

1.1 Purpose

The purpose of this document is to describe the participant's XML-based programmatic interface to the PJM RPM system.

The participant uses this document to guide in the formulation of messages to be sent to the RPM system and the interpretation of messages received from the RPM system. This document describes the addressing, exchange protocol, data, and XML formulation for all defined messages.

The reader of this document is assumed to be a software engineer whose intent is to understand the requirements of the participant's interface to the RPM system and to implement the software necessary to exchange data.

The business market rules that govern the data exchange for the RPM system are not described in this document. Market rules are published by the PJM separately (see reference [1]).

1.2 Technology Prerequisite Knowledge

In order to design and implement the XML data exchange software interface to the PJM RPM system using this programmatic API, the reader should be familiar with:

- Using XML for data description, XML Schemas, XML Namespaces, and XML parsing and validation.
- Protocols HTTP, HTTPS, and TCP/IP.
- Security and authentication technologies: encryption, authorization, and SSL (secure sockets layer).
- General network communication software methodology.

1.3 Document References

The following documents are pre-requisite reading material supporting this document.

- [1] [PJM RPM Business Rules Revised](#), published by PJM. Document identifier # 382144-v32 Version 10.2, Revised 04/17/2007.

The following references are suggested helps for the technology used by the programmatic API.

[2] Professional XML, 2000 by a myriad of authors, published by Wrox Press.

There is also a Beginning XML by the same publisher but this edition here is more complete.

[3] Professional XML Schemas, 2001 by a countable set of authors, published by Wrox Press.

This book also has sufficient discussion of XML namespaces to provide background for the Programmatic API.

[4] Professional Java Web Services, 2001 by a number of people, published by Wrox Press.

This reference has a good discussion of the SOAP standard giving a number of examples. The inclusion reference is not necessarily an endorsement of the Java language nor does it suggest that the programmatic API is a web services implementation.

[5] SSL and TLS Essentials, 2002 by Stephen Thomas, published by Wrox Press.

A good overview of the SSL protocol. However, it is recommended that SSL capability be acquired from a 3rd party vendor or the OpenSSL software be considered.

[6] RFC 2616: Hypertext Transfer Protocol, HTTP 1.1, 1999 by R. Fielding et' el, published by the Networks Working Group, IETF.

The best description of HTTP is the RFC documenting the standard.

1.4 Terminology

The following terms and acronyms are used by this document.

ACK – acronym meaning acknowledgement and indicates a normal and successful acknowledgement message. Contrast with NAK.

base64 – a coding scheme used to encode the username and password on the HTTP authorization header. The base64 encoding algorithm is published in several locations and defined by RFCs 1521 and 2045. Public available software (Java API, Visual Basic, and C) is available to code and decode base64 strings.

ComplexType – the type descriptor used in this document for XML complex type declarations. A complex type is a type that has sub-elements. Contrast this with a SingletonType.

CRLF – acronym meaning *C*arriage *R*eturn *L*ine *F*eed and is the definition of a line terminator. A line is contiguous set of octets (8-bit characters) that is terminated by a CRLF character. The CRLF character is often platform dependent and may be defined as the hex codes 0x0D, 0x0A or simply as the hex code 0x0A. The usage of 0x0D, 0x0A two-character couplet is standard for Windows platforms (e.g. Windows NT). The usage of 0x0A, single-character, is standard for many Unix platforms. The CRLF can be created using language features for generating a *new-line* character such as the C escape character `\n`.

Character Data – is the XML term that defines the non-markup data that exists between an XML start and end tag. For example, given a start and end tag of UNITMW, the character data in the following example is the number 400.0: `<UNITMW>400.0</UNITMW>`.

DTD – acronym meaning *D*ocument *T*ype *D*efinition and it is defined by the XML specification as the statements that describe the elements, attributes, and entities that comprise an XML document. This specification uses the XML Schema only (see entry for XML Schema). DTDs are not used.

First-Normal-Form – is the specification that character data defined by XML start and end tags (see definition of character data in this terminology section) represents a single data item. Data that does not fit the *First-Normal-Form* definition is data that includes sets of information, repeating groups, or structured elements. For the purposes of this specification, Date and Time data is considered to be First-Normal-Form data even though it defines sets of information (e.g. month, day, year, hour, minute).

HTTP – acronym meaning *H*yper-*t*ext *T*ransfer *P*rotocol. HTTP is an application-level protocol for distributed, collaborative, hypermedia, information systems. Or, more simply, HTTP is a network communications protocol used to send and receive data over the Internet. The version of HTTP used by this specification is 1.1 and this is often referred to as HTTP/1.1. The HTTP/1.1 protocol is defined by RFC 2616.

HTTPS – acronym meaning *H*yper-*t*ext *T*ransfer *P*rotocol *S*ecure. HTTPS is a secure protocol where the security is established by SSL (see terminology entry). HTTPS does not define nor does it add new communications features to HTTP, it is merely a secure version of HTTP. The HTTPS name is used to specify the protocol in the URL declaration to identify it as being secured by SSL.

IETF – acronym meaning Internet Engineering Task Force and it is the body whose members work together to define, specify, and regulate the various standards used for Internet network communications. The RFCs referenced by this specification are defined and maintained by the IETF. The IETF web site can be consulted for more information: www.ietf.org.

ISO – acronym meaning *I*nternational *S*tandards *O*rganization and it is the standards body responsible for a number of international standards used implicitly and explicitly by this document. Explicit standards cited by this specification include ISO 8601 used to define Date and Time standards and conventions; and, ISO-8859-1 used to define the XML operative character set. In the deregulated electricity industry, ISO also means Independent System Operator. That definition is not used in this document.

NAK – is the acronym meaning negative acknowledgement and is the opposite in meaning to the ACK acronym. NAK is commonly given when a network communications message is rejected or not processed completely. Contrast with ACK.

RFC – acronym meaning Requ^eFor Comments. The RFC is the documentation method used by the IETF for developing and documenting network communications standards for the Internet community. Several network communications standards cited by this specification are defined by RFCs. Each RFC is identified by a number. For a complete list of RFCs and to obtain copies of RFCs, see the information located at the IETF web site: www.ietf.org.

Request/Reply – is a messaging style where a client sends a request message to a server and the server responds with a reply message. Request/Reply is also sometimes called Client-Server messaging. In the case of the PJM RPM interface, the participant is always the client who initiates the request and PJM is always the server who responds with a reply.

RTO – Regional Transmission Organization.

SingletonType – is the type name used in this document to describe a single element definition that does not have any element sub-structure. It may define attributes that are used on the type and it may or may not define character data (note: the term *character data* in this sense is specific to XML terminology, see glossary entry).

SOAP – means Simple Object Access Protocol and it is the protocol used to *wrap* the various RPM data content messages described by this document. SOAP is used in a number of different messaging patterns. Various profiles of usage have been established and documented. The profile assumed by this RPM specification is to use SOAP as a message wrapper to provide common root context to make routing and handling of messages a little easier. SOAP is not used by this specification for its more popular profile: as a remote procedure call (RPC) mechanism.

SSL – acronym meaning Secure Sockets Layer. SSL is a protocol standard used to establish a secure, encrypted, connection between a given client and server. SSL Version 3.0 is the protocol used for establishing the secure communications between participants and PJM. SSL is an industry de-facto standard developed originally by the Netscape Corporation.

TCP/IP – acronym meaning Transport Control Protocol over Internet Protocol. TCP/IP is actually two separate protocol standards however, as used in this specification (and, in most other applications), TCP/IP are considered a single network communications protocol. TCP/IP is the protocol specified by various RFC documents published by IETF (RFCs in this case are not as useful as other more popular publications of the TCP/IP standard). TCP/IP is used as the carrier protocol for all messages defined by this specification.

URI – acronym meaning Universal Resource Identifier and is a similar construct to URL (defined next). URL in fact is a kind of URI. The URI is used to name Internet resources that are not necessarily Internet locations. More information on URI can be found in RFC 1630.

URL – acronym meaning Uniform Resource Locator. URL is the standard method for specifying network addresses and network resources used by Internet protocols. The URL defines the protocol, network host address, optional port number, resource path and fragments. URLs are used to specify PJM RPM server addresses that receive messages sent by the participants. URL is defined by RFC 1738.

Valid – as used in this context, *valid* is the measurement of an XML document that is correctly formatted according to its schema. PJM accepts only valid XML documents in messages received from participants. All valid XML documents are implicitly also *well-formed*.

W3C – is the acronym meaning World Wide Web consortium which is a Internet community standards body whose purpose is to develop, publish, and maintain standards for the Internet community. Since W3C is not a government body, its publications are called *recommendations* as the term *standard* is defined as government authorized. The XML and HTML recommendations (aka *standard*) is published by the W3C. More information on the W3C publications and organization can be found at their web site: www.w3c.org.

Well-formed – is used to measure an XML document that is formatted according to the syntax rules set forth by the XML recommendation. A *well-formed* document can be correctly parsed by an XML parser. A document that is not well-formed may fail to parse completely.

XML – is the acronym meaning Extensible Markup Language. XML is a W3C published recommendation. XML is used as the basic format method for all messages defined by this specification. More information on XML can be found at the web site: www.w3c.org/xml.

XML Namespaces – provide a mechanism for qualifying the name of an XML coded entity per a unique identifier defined by a URI. The use of namespaces allows element and other tag names to be derived from more than one vocabulary without conflict or collision. Namespaces, though optional with many XML usage patterns is required for this RPM specification. For more information, consult the web site www.w3c.org.

XML Schema – provided an improved means of declaring structure and content of an XML document. XML Schema is a replacement for the older document type declaration specified by the DTD. Use of XML Schemas are required by this RPM specification. For more information, consult the section on XML Schema at the web site www.w3c.org or references [2] and [3] cited in section 1.3 of this document.

2 Introduction to the Programmatic API

The programmatic API is an XML based messaging protocol used for the exchange of messages between market participants and PJM. The XML message is formulated as a SOAP wrapped payload carried by the HTTP protocol.

The participant software must implement the messaging protocol according to the specification described in this document. This software implementation relies heavily on standard technologies that can be obtained freely on the Internet (i.e. open software) or that can be obtained from 3rd party vendors.

2.1 Messaging Overview

The programmatic API is a messaging API used by PJM RPM to exchange data with the RPM market participants. In function, the programmatic API mimics the interactive web user interface: most functions supported by the web pages are supported by the programmatic API.

2.1.1 The Role of the Participant

The participant uses the programmatic API to implement an automated interface to the RPM market services provided by PJM. Using the programmatic API, the participant can implement software that integrates efficiently with other applications and processes on the participant side of the network. This allows the necessary message exchange to fit more easily into the participant's business processes and methods.

The participant uses the programmatic API to:

- Submit resource offers into the base residual and incremental auctions.
- Submit daily load contribution information.

2.1.2 The Role of PJM

PJM provides services that support the programmatic API. These services are identified by URLs (Universal Resource Locator) and made available to the participant. When the PJM server receives a request at a given URL, the following processing is performed:

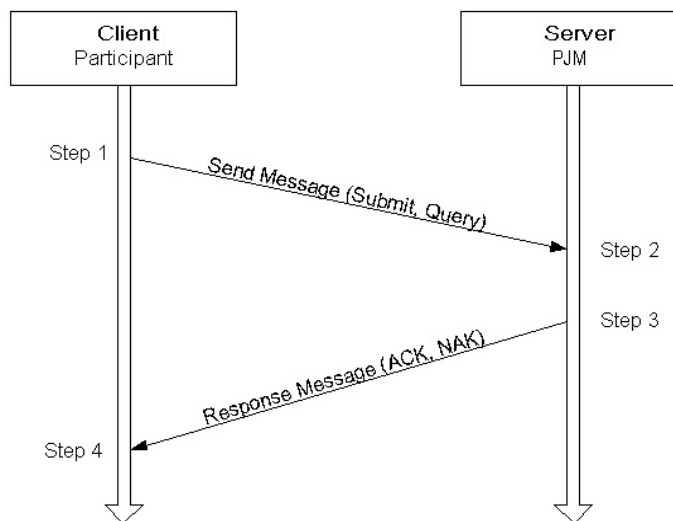
- The request is validated both syntactically and semantically. Any error results in an error response.
- If the request is a data submission then the data is validated per the RPM database constraints and per the business rules. Any violations result in an error response and the total rejection of all data composing the message. If there are no errors, then the data is entered into the RPM database.

- If the request is a data query then the message is dispatched to the known service handler for the requested packet. If the request is correctly formatted then the data requested is packaged as a SOAP wrapped XML payload and returned to the participant.

2.2 Request/Reply Messaging

The messaging protocol is called request/reply (or, request/response). The client issues a request to the PJM server and a reply (response) is returned by the server to the client. The client initiates all request/reply messages.

The following diagram illustrates the request/reply protocol.



The client is the market participant that is sending a request to the PJM server handling the message. The processing steps are outlined below:

Step 1 – the client software formulates the request message and sends it to the server. Request messages may be to submit bids and offers or query the RPM market database. After sending the request, the client software waits for the response (step 4). Because the client software initiates the request and waits for the response, this is considered a *synchronous* messaging protocol.

Step 2 – the server software is actively listening for new messages and receives the message sent by the client. The server software processes the message which includes validating the message request, processing message action, and formulating the message response (step 3).

Step 3 – the server formulates the response as either an ACK (successful request) or a NAK (unsuccessful request resulting in error response). The successful request may be a simple indicator of the success of the requested action or it may also include a data packet returned to the client as the response to the requested action (e.g. a data query).

The unsuccessful response will always include an error packet that identifies each of the errors encountered in the request.

Step 4 – the client software receives the response message and acts accordingly depending on whether the response indicates success (ACK) or failure (NAK).

The PJM RPM server implements a message listener using a standard HTTP webserver that supports SSL for encryption and authentication¹. Therefore, submitting a message request is tantamount to submitting an Internet web page request; although, instead of using the HTTP GET command, the HTTP POST command is used.

Therefore, the participant client must implement software that makes an HTTP connection using SSL along with the authentication scheme (username/password credentials). Once the HTTP connection is established the message is sent as a POST command to the server.

2.3 Web Technologies

As evident by the previous discussion, the programmatic API is implemented using standard *web* technologies. In particular, the following technologies are used by this scheme:

- XML, Extensible Markup Language
- SOAP, a web services API
- HTTP/HTTPS
- SSL and Authentication

2.3.1 XML – Extensible Markup Language

XML means eXtensible Markup Language. It is a structured, hierarchical, tag based coding language that is very suitable for describing data that is sent over the public Internet. There are many good references for learning about XML and a few are listed in section 1.3 ([2], [3]) of this document. The specification and details of XML will not be repeated here. However, a few comments on “why” and “how” XML is used is discussed briefly below.

¹ All message exchange between the participant and the PJM server uses HTTPS protocol which is HTTP over an SSL (encrypted) session. The name HTTP names the protocol commands, the suffix of S refers to the secure connection. When discussing the protocol, the name HTTP will be used.

XML Versus CSV

XML is a replacement for data file exchange using CSV format. CSV means *comma separated values* and refers to a commonly used ASCII coded file format where each line is a set of data values separated by a comma (or, some other field separator character such as a TAB). Each line typically refers to a set of associated data and given a particular interpretation by the producers and consumers of the data. The interpretation is usually ruled and defined by an external element that is often cryptic and obscure.

XML has the following advantages over CSV:

- XML is usually easily readable where each data value is described by a name. CSV often does not describe data by name.
- XML supports structure allowing nested or hierarchical relationships that can be validated for correctness. CSV is necessarily flat in structure often leading to the need for repeating groups of data to model data hierarchy.
- XML supports strong data typing (using XMLSchema language) that can be validated as part of the XML parsing process. Since parsers are standard, given the XMLSchema, an XML message can be parsed and validated easily by anyone. Custom validation software is not required (for example, standard tools such as XMLSpy² can be used to validate a message against the schema).
- XML supports data transparency where character data can be composed of any character code (assuming a few proper escapes where needed by the language). CSV on the other hand has trouble with embedded commas in character data if the comma is a field separator.

XML Schema

XML Schema is a relatively recent addition to the XML family of standards. XML Schema is a structured language used to describe other XML documents. XML Schema is itself an XML document described by an XML Schema.

XML Schemas are a replacement for the XML DTD (document type definition) which is still commonly used to describe XML. A DTD though lacks many of the features of XML Schema, such as strong data typing, name space management (DTDs do not support name spaces). XML Schema also supports a more powerful relationship and structure specification.

XML Schema is used to describe all RPM data exchange messages. DTDs are not used and they are not supported. There are many references describing XML Schema and how it is used. Those references already listed for XML ([2], [3]).

² XMLSpy is a third-party utility product used to create and validate XML schema and XML instance documents. XMLSpy is recommended as a tool for anyone using XML and is available from ALTOVA (see www.xmlspy.com or www.altova.com).

XML Namespaces

XML Namespaces are used to support the association of specific XML markup to a particular vocabulary and schema. Namespaces allow the mixture of XML markup from different vocabularies. In fact, using SOAP and XML Schemas require the use of namespaces and at least two vocabularies.

An XML Namespace is a unique identifier specified using a URI (uniform resource identifier). Thus, a namespace identifier looks like a URL or a web page address. However, an XML Namespace identifier is not a web page address and there is no expectation of finding any resource at the URI that is used. The combination of the unique domain name and the URI path establish the uniqueness that is required of namespace identifiers.

The use of XML Namespaces as shown later in this document are required. Failure to use the proper namespace will result in an XML validation failure and an error response. More information on XML Namespaces can be found in the XML references already cited.

2.3.2 SOAP – A Web Services API

SOAP means Simple Object Access Protocol. It is based on XML and is designed for the exchange of information in a distributed computing environment. The SOAP protocol was originally designed for RPC (remote procedure call) style of computing but it also supports other message exchange profiles. SOAP is not used as an RPC by the PJM RPM system; rather, it is used to wrap message payload in a common, standard, way to facilitate handling, routing, and common processing.

The SOAP protocol is an outside wrapper called the Envelope. This Envelope encloses all other XML data. The Envelope itself contains two distinct structures: Header and Body. The Header element is not used by this specification (it is considered optional by SOAP). The Body element contains the entire payload.

The SOAP standard used by this specification is SOAP 1.1. There are a growing number of good references for SOAP, the one provided here (reference [4]) is just one recommendation.

Web-Services

Web Services is a defined set of technologies, tools, and standards (W3C recommendations) used to support a general service layer of information exchange. A formal Web Services application is described by a Web Services Description Language (WSDL) configuration file and used by some software tools for supporting a service connection.

SOAP is a protocol that is also defined as part of the Web Services standards. The W3C organization has published standards for the Web Services Description Language that is used in concert with SOAP and other technologies. However, this RPM application *is not* a formal Web Services application. A Web Services Description Language interface is not supported at this time.

2.3.3 HTTP/HTTPS

HTTP means Hyper-text Transport Protocol and HTTPS means Hyper-text Transport Protocol Secure. HTTP is the transport protocol used by the programmatic API (this specification) to carry the RPM XML messages over the Internet. HTTPS is the same protocol as HTTP except that it is a secure, encrypted session established by SSL. All the commands, headers, error responses, and body format for HTTPS is the same as HTTP. Therefore, in this and many documents, HTTP is generally used to refer to both HTTP and HTTPS with regard to the actual protocol rules. The HTTP standard is specified by RFC 2616 (reference [6]). HTTP 1.1 is used by this specification.

Although HTTP is the protocol for message exchange using the programmatic API, it is not the only method supported. File transfer is also supported for download. The File transfer format and naming rules is described later in this document.

2.3.4 SSL and Authentication

SSL means Secure Socket Layer and it is a protocol used to establish a secure, encrypted, TCP/IP session between the client and server computers. SSL is used for all data exchange using this programmatic API. SSL is not the authentication method though -- it merely provides the security for the authentication means.

SSL is an industry defacto standard developed by Netscape Corporation and it is the most commonly used and supported encryption protocol. Other protocols exist. The standard TLS is an IETF endorsed standard to replace SSL and thus is compatible with SSL. TLS means Transport Layer Security. SSL is referenced instead of TLS because it is still the industry dominant protocol used for establishing an encryption channel between client and server.

Authentication is the establishment and verification of the user submitting the requests to the PJM Server. Authentication is based on BASIC realm username and password authentication using the HTTP authorization header (explained in more detail later). The username and password are granted and managed by the PJM eSuite system. All users must be registered with eSuite to be authenticated for PJM RPM operations.

3 Common Principles

This chapter describes common requirements and messaging concepts used by all message types.

3.1 RPM Participant Registration

All RPM participants using the XML programmatic interface must be registered with PJM's eSuite system. Each user must have:

- A username that is specified on the authorization header of each HTTP message.
- A password that is specified on the authorization header of each HTTP message.
- Permission grant to access RPM public and private information. Authorization permissions are granted for: RPM public, RPM private read-only, and RPM private read and write.

3.2 RPM Server Addressing

All functions and data provided by the RPM Server are addressed using a Uniform Resource Locator (URL). Several URLs are defined as shown in the following table. Each message sent to the RPM server must use the appropriate URL. The appropriate URL is defined for each message type defined later in this specification.

Note: the actual host computer name shown below (*rpm.pjm.com*) is subject to change. Also, there may be more than one host supported by PJM for support of various test and sandbox functions.

Function	URL
Query Public or Private Data	https://rpm.pjm.com/erpm/services/query
Submit Private Data	https://rpm.pjm.com/erpm/services/submit

3.3 HTTP Command and Headers

All messages submitted to the PJM Server and all messages returned to the participant as responses are coded as HTTP POST command messages.

3.3.1 HTTP Request Message Format

The following example highlights the required format and content of the request message for both data submit and data query (although this example shows a query on private data as shown by the URI).

```
POST /rpm/xml/query HTTP/1.1
Host: rpm.pjm.com
Authorization: BASIC kjfpekmd3kjkj=
Content-Type: text/xml; charset="UTF-8"
Content-Length: xxx
SOAPAction: "/rpm/xml/query"

<body content>
```

In this HTTP POST command example, the SOAP wrapped XML message (described below) would appear where the <body content> is shown.

The SOAPAction HTTP header describes the SOAP action or method to execute. This is included here but it is not used by the PJM RPM Server software³. The SOAPAction header maybe included by the participant but it is ignored by the PJM RPM server.

That gobbledegook string on the Authorization header represents the base64 coded username and password. The authorization realm required by this specification is the BASIC scheme as shown above on the Authorization header. The base64 encoded string is a coding of the username and password in the format of: *username:password*. That is, the username, followed by the : character, followed by the password. This string is then base64 encoded and specified on the Authorization header⁴. This header is required for all requests.

3.3.2 HTTP Response (reply) Message Format

The following example highlights the content of a typical response message.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: xxx

<body content>
```

The interpretation of the above lines is standard HTTP. The 200 status code indicates success. All responses returned by the PJM RPM system are considered successful responses using a 200 status code even if they are reporting an error in the content of the message. If an error should be raised by the PJM web server or if there is an authorization failure or bad URL then other HTTP response codes can be returned. Codes that may be returned include:

³ The SOAPAction header is a requirement of SOAP 1.1, the current version of the SOAP standard but it is expected to be removed or made optional in SOAP 1.2.

⁴ The base64 encoding schema simple and well documented by RFC 1421. Utilities for performing base64 encoding and decoding are freely available on the Internet (just do a google search on base64).

400 – bad request is returned if there is a formatting error in the headers or the HTTP command line.

401 – Unauthorized is returned if the request does not include an authorization header or if the authentication fails to authorize the given username and password.

404 – URI not found is returned if the URL specified on the request is unknown.

405 – Method not allowed is returned if any request to the specified URLs given any other command beside POST. Only the POST method is allowed.

500 – Internal Server Error is returned if some part of the server fails to respond or has crashed. See comment below on SOAP and the 500 error code.

It is possible that other error codes would be received but these are the most common likely to be seen during actual operation.

The SOAP standard requires that error code 500 be used in conjunction with any error message returned using the SOAP Fault response message. The main purpose of the SOAP Fault response message is to report failures due to exceptions, argument data type errors, and other RPC oriented problems. The PJM RPM implementation using SOAP is a message based request/reply exchange that does not suffer from the typical RPC type error handling and exception mechanism. Therefore, the SOAP Fault response message is not used and error code 500 is not reported from the PJM RPM server software.

3.4 SOAP Envelope

In the above HTTP request and response examples, the <body content> refers to the SOAP envelope. The SOAP envelope wraps all message content including query requests, query response, submittal request, submittal response, and error response.

The following example shows the canonical SOAP envelope used for all request and response messages. The HTTP header lines are representative of the HTTP structure (for a response message in this example), the body of the SOAP envelope contains the actual message using the following elements:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
<env:Header/>
<env:Body>

    --- actual message body goes here ---

</env:Body>
</env:Envelope>
```

There are other parts to the SOAP standard that are not shown by this example. For example, encoding style can be specified, other name spaces can be referenced.

The SOAP Header which is optional and may appear before the SOAP Body is not used. The Header element can be specified as an empty tag, as shown above and other parts of this document or it can be ignored on submission of data or query requests. The empty Header element is always included on XML response messages. These other parts to the SOAP schema are not used at this time by PJM RPM.

The SOAP Body element contains the XML Query Request or the XML Submit Request elements as documented in later chapters describing individual query and submit messages. When these elements are specified, it is recommended that the namespace required by the PJM RPM be specified as a global namespace operating from that position. For example,

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Header/>
<env:Body>
  <SubmitRequest xmlns="http://erpm.pjm.com/rpm/xml">
    ...other data...
  </SubmitRequest>
</env:Body>
</env:Envelope>
```

3.5 XML Namespaces

Two namespaces are assumed and used by this specification. One namespace is used by SOAP and the other name space is used by the RPM XML Schema. These namespaces are:

SOAP:	http://schemas.xmlsoap.org/soap/envelope/
PJM XML:	http://erpm.pjm.com/rpm/xml

3.6 XML Schema

Two XML Schema references are required to successfully parse and validate the XML messages described by this document. One schema is for the SOAP 1.1 standard and the other schema is for the PJM RPM XML messages. They are:

SOAP:	<code>http://schemas.xmlsoap.org/soap/envelope/</code>
PJM XML:	<code>http://erpm.pjm.com/rpm/xml/schema</code>

The schema should never be located using the schemaLocation element inside of an XML message. All schemas used by the PJM RPM server are provided externally. The message content schema, if specified, is never used. **Note that the PJM XML Schema location is subject to change. Please contact PJM for current URL used to locate the PJM XML Schema.**

3.7 Error Response

There are three kinds of response messages returned to the client: success response, success with data response, and error response. The individual message success and success with data responses are described in later sections. The error response is described in this section and is common to all request type messages.

The error response is given for the following types of problems:

- Malformed XML format, that is, unable to parse the XML correctly.
- Invalid XML message, that is, XML content does not validate against the schema. This includes many of the data type, range, relationship errors that can be found.
- Violation of business rules include errors such as market is not open, invalid names or data values, and so on.

When an error response is returned, the transaction request is rejected and not executed and the market database is not modified at all. For example, if a data submit message has an error in just one part of the overall message, the entire message and all of its data is rejected. No partial data submits are allowed or supported. If a query request has an error in any part of the request then no data is returned to the client even though other parts may be error free.

Note that if any of the HTTP errors are returned (as described previously) then there is no body returned with the message that is meaningful to this programmatic API (it might be some other body provided by an intermediary component that operates outside of this specification and therefore may be useful). All error responses described in this section are returned in a successful (HTTP/1.1 200 OK) message.

The error response message has the following SOAP wrapped XML format.

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Header/>
<env:Body>
  <SubmitResponse xmlns="http://erpm.pjm.com/rpm/xml">
    <Error>
      <Code>ORA-00000</Code>
    </Error>
  </SubmitResponse>
</env:Body>
</env:Envelope>
```

```

    <Text>Market is not open</Text>
    <Line>342</Line>
  </Error>
  ...
</SubmitResponse>
</env:Body>
</env:Envelope>

```

<Error> is used to specify a single error occurrence, there may be multiple Error elements returned in a single error response message.

<Code> is used to specify the error code (if one exists). The Code element is an optional part of the Error. Error codes may refer to specific software modules used by PJM RPM. The error codes are used for communication between participants and PJM when a particular problem occurs. In general, the error code is an abbreviation of the text message in <Text>.

<Text> is used to contain the error message text describing the problem. This is a required element and always returned.

<Line> is an optional line number that can be returned specifying the line where the problem occurred. Not all submittal methods use the notion of a line so this value is not always available and it may not be useful. It depends on the nature of the submitted data.

3.8 XML Data Type and Specification Semantics

In general, data specified using the XML Schema rules is bound by very tight data type and constraining facets. However, some fields require special care in how they are specified since the user has complete freedom to specify any characters he wishes and some of these cause problems for XML or have dubious interpretations.

3.8.1 Character Case

- Character case, upper, lower, and camel case must be honored by all XML specified in this document. Within the body of the XML, there are no case insensitive strings. Case must be honored at all times.

3.8.2 XML Boolean Data

Boolean data is a recognized type defined by the XML Schema. Boolean is binary data represented as true or false. The Boolean values acceptable for true and false are shown in the table below:

State	XML Representations
True	true 1

False	false 0
-------	------------

For example, if a Boolean element called <Veracity> were specified as input, each of the examples below would represent a true value being submitted:

<Veracity>true</Veracity>

<Veracity>1</Veracity>

3.8.3 XML Date and Time

All date and time values must follow the XML Schema data type rules for date and time. The formats used by this specification include:

date	YYYY-MM-DD (example: 2003-02-03)
dateTime	YYYY-MM-DDTHH:MM:SS (example: 2003-02-03T12:30:00)
dateTime with Zone	YYYY-MM-DDTHH:MM:SS.sss+HH:MM (example: 2003-02-03T12:30:00.000-05:00)

The first DateTime format shown above is the default format used for any input operations. Currently, there are no requirements to input a date and time field. The second format, showing the time zone is used on output, such as a transaction timestamp.

The XML Schema data type description as documented by the W3C standards group should be consulted for the exact representation and handling of the dateTime data type. Briefly, the above fields are interpreted in the following manner:

YYYY	The year as a 4 digit number such as 1999, 2000, 2001, and so on. Abbreviations using 2 digits are not allowed.
MM	The month as a 2 digit number ranging from 01 through 12. January is represented by 01, February by 02, and so on. Leading 0 for months 01 through 09 must be specified.
DD	The day of the month as a 2 digit number ranging from 01 through 31. The leading 0 for days 01 through 09 must be specified.
T	A literal "T" value that acts as the separator between calendar day and the clock time values.

HH	The hour of the day as a 24 hour clock using 2-digits from 00 through 23. Leading 0 must be specified for hours 00 through 09.
MM	The minute of the hour as a 2-digit value ranging from 00 through 59. Leading 0 must be specified for minutes 00 through 09.
SS SS.sss	The second of the minute as a 2-digit value ranging from 00 through 60. Leading 0 must be specified for seconds 00 through 09. The second form includes the fractions of a second shown as SS.sss where the .sss fractional part is any fractional value of arbitrary precision. The value used with this specification on output is .000. Note that the meaning of 60 seconds or more is useful for special leap second days of the year and this feature is not used by this specification.
+	This field represents a + (plus) or a – (minus) sign to separate the time zone offset from GMT. A minus sign is used for those times west of the prime meridian and therefore lag GMT time.
HH:MM	This field to the right of the + or – sign of the time zone offset indicates the time zone offset in hours. The value is represented as whole number of hours in this specification (that is, the MM field is zero).

The date and time format rules above are part of the XML standard. However, this specification will use .000 for the sub-second offset and for time zone only whole hours are used.

All time zones default to Eastern Prevailing Time.

3.8.4 XML Character Data and Markup and Escaped Characters

XML text consists of intermingled character data and markup. Markup takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, and CDATA section delimiters. Anything that is not markup is called *character data* of the XML document.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form only when used as markup. These characters cannot appear in their literal form within *character data*.

If these characters are needed within character data, they must be escaped using either numeric character references or the strings “&” and “<” respectively for & and <. The right angle bracket may appear in character data in some circumstances but it is best to escape this character using “>” or the numeric escape equivalent.

For example, if an element calls for the specification of character data for a name for instance, that name cannot include any of the characters mentioned above. Consider the following:

<pre><name>MW&MVAR</name></pre>	<p>This is illegal because the & sign must be escaped. The result is that the parser will see the & as a escape which means that the next few characters are sucked in and interpreted as special characters. The result is that the < and the / are eaten and the resulting XML document is not well-formed.</p>
<pre><name>MW&amp ;MVAR</name></pre>	<p>This is the correct specification when the & sign is needed.</p>
<pre><name>MW and MVAR</name></pre>	<p>This is better yet, avoid the & character altogether.</p>

The XML specification should be consulted for a more detailed explanation of the differences in handling character data, markup, entities, entity references and so on. Our motivation though is to create an XML Schema that will not result in any of these situations however that assumes that the user always does the right thing. This explanation is for those users who get caught doing the wrong thing.

3.8.5 XML Null Data

Sometimes it might be useful to submit a null value into the database⁵. This is allowed for only certain data fields defined by the XML Schema but the method of submitting a null is to submit an empty element. For example, the following two lines specify an empty element:

```
<Price/>

<Price></Price>
```

These two examples are considered equivalent, the result is the same. They specify an element by name, Price, but do not specify any character data. If the XML Schema defines non-null character data than this representation is interpreted to submit a null value. This null value overwrites any existing value in the database for that field with a null.

Constrast this handling of null data with an absent element described next.

3.8.6 XML Absent Elements

Some elements are optional. They may appear in the XML document or they may not. If they do not appear, the default handling is usually governed either by the XML Schema definition or the

⁵ At this time, the message format do not define any data fields where a null value can be entered.

application. If the XML Schema specifies a default value than that value is substituted just as if the element were specified with that value.

If there is no default value associated with the element than the application treats this as if the element did not exist. That is, if the operation is some kind of replace of existing data and the database field reflecting the missing element is left alone. It is not modified in any way. It is not set to null. If there is an existing value, it is retained.

3.9 File Upload and Download Format

The file upload and download XML format is identical to the message body of the HTTP query request, submit request, or response. The only missing lines are those specifying HTTP headers. The following example shows a query response as an HTTP message and the same query response formatted as a file.

First, the HTTP message query response⁶:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
<env:Header/>
<env:Body>
  <QueryResponse xmlns:mkt="http://erpm.pjm.com/rpm/xml" >
    ...more data...
  </QueryResponse>
</env:Body>
</env:Envelope>
```

The equivalent file download format appears as shown below:

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
<env:Header/>
<env:Body>
  <QueryResponse xmlns:mkt="http://erpm.pjm.com/rpm/xml" >
    ...more data...
  </QueryResponse>
</env:Body>
</env:Envelope>
```

Line breaks may be used to format the response message for readability and processing but are not part of the message content. There is no meaning to a line break in the XML message. Also, all line breaks are issued as single byte LF codes (the typical Unix style of line break, the Microsoft Windows style of CRLF are not used).

The file is named according to the type of data but this name can easily be changed by the user as part of the download process therefore not much substance is given to the file names.

⁶ XML example is shown for illustrative purposes only, it does not necessarily refer to an actual message with realistic data.

Therefore, to submit data for upload, the very first line in your message is the `<?xml version="1.0"?>` line following by the SOAP envelope and the content of the Body.

4 Submit Request

The submit request is used to send data to the PJM RPM server and into the RPM market database for further processing. Data submitted is defined for the following purposes:

- Submitting or updating offers and/or bids for the base and incremental auctions associated with RPM markets.
- Submitting or updating load contribution data.

All data submit actions are executed under transaction control. All data submitted is considered private, although some submitted is viewable by more than one participant, i.e. transactions and load contribution data.

The data submit is initiated using a data submit request element as shown below. This submit request is the payload that is enclosed by the SOAP Body element. Only one such submit request element can be specified. The SubmitRequest may contain one or more instances of individual data elements containing the submitted data. By definition, only one kind of data submit is possible for each message.

Example of submit of an offer to a market:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
  <SubmitRequest xmlns="http://erpm.pjm.com/rpm/xml">
    <ResourceOfferSet CompanyName="XYZ">
      ...more data...
    </ResourceOfferSet>
  </SubmitRequest>
</env:Body>
</env:Envelope>
```

The response message is always returned as a SOAP wrapped payload indicating either success or error. The success indicator is the same for all types of data submit. The successful submit always has the same format as shown below:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
  <SubmitResponse xmlns="http://erpm.pjm.com/rpm/xml">
    <Success>
      <Message>Submit Successful</Message>
    </Success>
  </SubmitResponse>
</env:Body>
</env:Envelope>
```

The format for each of the submit requests follows the same naming conventions.

The following example shows an error response as a result of some problems in the submit:

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
  <SubmitResponse xmlns="http://erpm.pjm.com/rpm/xml">
    <Error>
      <Code>BlueGreen</Code>
      <Text>Market does not exist</Text>
    </Error>
  </SubmitResponse>
</env:Body>
</env:Envelope>

```

All private submit requests must be issued to the following URL:

```
https://rpm.pjm.com/rpm/xml/submit
```

If you choose to specify a SOAPAction HTTP header, then it must have the following value:

```
/rpm/xml/submit
```

Each of the following sections describes the defined submit requests.

4.1 Submitting a Resource Offer into an Auction

4.1.1 Purpose

This message format is used to submit resource offers into a base residual or incremental auction. In order to submit a resource offer the market must be open. Prior to market close and clearing, RPM offers can be modified.

4.1.2 Message Format

The full resource offer message format is described below.

```

<SubmitRequest>
  <ResourceOfferSet CompanyName="CompanyXYZ"
    PlanningPeriodName="2009/2010" AuctionType = "BASE">
    <ResourceOfferSetDetail>
      <ResourceName>ResourceXYZ</ResourceName>
      <ResourceType>GEN</ResourceType>
      <ResourceOffer>
        <EFORd>0.001</EFORd>
        <NewUnitPricing>>false</NewUnitPricing>
        <CapBlocks>
          <CapacityBlocks id="1">
            <MinMW>0.0</MinMW>
            <MaxMW>15.0</MaxMW>
            <Price>0.00</Price>
            <SelfSchedulingOption>Regular
Schedule</SelfSchedulingOption>

```

```

    </CapacityBlocks>
  </CapBlocks>
</ResourceOffer>
</ResourceOfferSetDetail>
</SubmitRequest>

```

The following table describes each of the elements and attributes and how they are used:

Element or Attribute	Data Type	Description
<SubmitRequest>	ComplexType	The root element for all submit requests. Occurs just once and placed within the SOAP envelope Body element.
<ResourceOfferSet>	Complex Type	
<ResourceOfferSetDetail>	Complex Type	
<ResourceName>	Character	Name of Resource
<ResourceType>	Character	GEN or DEMAND
<ResourceOffer>	Complex Type	
<EFORd>	Number(6,5)	
<NewUnitPricing>	Boolean	
<RMBlocks>	Complex Type	EFORd Segment
<Blocks>	Complex Type	
id	String	Identifier of EFORd segment. Domain is 1.
<MinMW>	Number(8,1)	
<MaxMW>	Number(8,1)	
<Price>	Number(10,2)	
<CapBlocks>	Complex Type	

Element or Attribute	Data Type	Description
<CapacityBlocks>	Complex Type	
id	String	Identifier of segment. Domain is 1-10.
<MinMW>	Number(8,1)	
<MaxMW>	Number(8,1)	
<Price>	Number(10,2)	
<SelfSchedulingOption>	Character	Regular Schedule, Self Schedule, Flexible Self Schedule

4.1.3 Response Message

The Success (ACK) response message is described below.

```
<SubmitResponse>
  <Success>
    <Message>xxx</Message>
  </Success>
</SubmitResponse>
```

The following table describes each of the elements and attributes and how they are used:

Element or Attribute	Data Type	Description
<SubmitResponse>	Complex Type	The root element of all submit response elements. Occurs just once.
<Success>	Complex Type	The element indicating a successful submit operation. Occurs just once.
<Message>	Character	Always returned.

The Unsuccessful (NAK) response is an error report as described in Chapter 4 {Error Response} of this specification. Errors that may be reported include:

- Invalid or improper XML
- Market does not exist.

- Market is not open.
- Violation of market rules.

4.2 Submitting Load Contribution Data

4.2.1 Purpose

This message format is used by an EDC to submit load contribution for an LSE in a zone/area up to 36 hours before the operating day.

4.2.2 Message Format

The full resource offer message format is described below.

```
<SubmitRequest>
  <LoadContributionSet CompanyName="COMPANYXYZ" >
    <load_contribution>
      <load_type>PEAKLOAD</load_type>
      <zone>ZoneXYZ</zone>
      <area>AreaXYZ</area>
      <lse>LSEXYZ</lse>
      <mw_amount>0.0</mw_amount>
      <effective_date>2007-06-01</effective_date>
    </load_contribution>
  </LoadContributionSet>
</SubmitRequest>
```

The following table describes each of the elements and attributes and how they are used:

Element or Attribute	Data Type	Description
<SubmitRequest>	ComplexType	The root element for all submit requests. Occurs just once and placed within the SOAP envelope Body element.
<LoadContributionSet>	Complex Type	
CompanyName	Character	Name of the EDC submitting the contribution data.
<load_contribution>	Complex Type	
<load_type>	String	PEAKLOAD or NSPL

Element or Attribute	Data Type	Description
<zone>	Character	
<area>	Character	
<lse>	Character	
<mw_amount>	Number(8,1)	
<effective_date>	Date	

4.2.3 Response Message

The Success (ACK) response message is described below.

```
<SubmitResponse>
  <Success>
    <Message>xxx</Message>
  </Success>
</SubmitResponse>
```

The following table describes each of the elements and attributes and how they are used:

Element or Attribute	Data Type	Description
<SubmitResponse>	Complex Type	The root element of all submit response elements. Occurs just once.
<Success>	Complex Type	The element indicating a successful submit operation. Occurs just once.
<Message>	Character	Always returned.

The Unsuccessful (NAK) response is an error report as described in Chapter 4 {Error Response} of this specification. Errors that may be reported include:

- Invalid or improper XML
- Market does not exist.
- Market is not open.
- Violation of market rules.

5 Error Response

The error response message format is described below:

```
<SubmitResponse> or <QueryResponse>
  <Error>
    <Code>xxx</Code>
    <Text>xxx</Text>
    <Line>xxx</Line>
  </Error>
</SubmitReponse> or </QueryResponse>
```

The following table describes each of the elements and attributes and how they are used:

Element or Attribute	Data Type	Description
<Error>	Complex Type	Specifies a single error report. Occurs 1 to many times (hopefully not too many times).
<Code>	Character	Optional error code. Error codes may be associated with various software modules. Error codes have meaning only when reporting problems to PJM or requesting help on a given error type. Occurs at most once. Only occurs if vendor specific error code exists (e.g. Oracle, Weblogic, etc.).
<Text>	Character	Specifies the text of the error message. Occurs 1 to many times. Normally, occurs just once, sometimes two or three <Text> elements are used per error.
<Line>	Character	Optional line number indicator provided only when it is available and when it has meaning. Occurs 0 or 1 times.