INTERNET PARTS ORDERING

Technical Implementation Guide
Version 3.0 Specification
# Table of Contents

- **Introduction** .................................................................................................................. 3
  - Purpose of this document .................................................................................................. 3
  - Limitations .......................................................................................................................... 3
  - Development of the IPO v3 Specification ......................................................................... 3
  - Related documents .............................................................................................................. 4
  - How this Document is Organized .................................................................................... 5
- **Technical Background** .................................................................................................... 5
  - Standards Context .............................................................................................................. 5
  - Participants Roles .............................................................................................................. 7
  - Public & Private Processes ............................................................................................... 8
  - Business Message Standards .......................................................................................... 8
  - From Document Exchange to Service Oriented Architecture ......................................... 10
  - Web Services Context ..................................................................................................... 11
  - Interaction Styles ........................................................................................................... 12
  - Web Services Interoperability Organization (WS-I) ......................................................... 21
  - SOAP Messaging Overview ............................................................................................ 21
  - WSDL .................................................................................................................................. 24
  - Transport Protocols ......................................................................................................... 25
  - SOAP Faults ..................................................................................................................... 26
  - Security .............................................................................................................................. 26
  - Compliance Summary ...................................................................................................... 27
- **Implementation Guidelines for Buyers** ......................................................................... 28
Planning ..................................................................................................................... 28
Implementing Synchronous Interaction ...................................................................... 29
Asynchronous Buyers ................................................................................................ 32
Implementation Guide for Sellers ................................................................................ 35
Overview ..................................................................................................................... 35
WSDL Usage ............................................................................................................. 35
Publishing Your IPO Web Service ............................................................................. 37
Creating the Buyer Service Client .............................................................................. 38
Security ...................................................................................................................... 38
Working with the IPO Distribution .............................................................................. 38
Overview ..................................................................................................................... 38
Getting Started .......................................................................................................... 39
Generating Code ........................................................................................................ 39
Additional Constraints ............................................................................................... 39
Implementation Alternatives ....................................................................................... 41
Overview ..................................................................................................................... 41
Microsoft .Net Implementation .................................................................................... 41
J2EE Implementation .................................................................................................. 41
EAI Implementations .................................................................................................. 42

Notice

The Auto Care Association, formally Automotive Aftermarket Industry Association (AAIA), makes no warranty of any kind regarding this material, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. Auto Care Association shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

©Copyright 2003 - 2014 Auto Care Association

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of the Auto Care Association. The information contained in this document is subject to change without notice.
Introduction

Welcome to Internet Parts Ordering (IPO). By implementing the standards contained in this document, your company will join a broad industry coalition that is working to improve the speed and efficiency of the automotive aftermarket.

Purpose of this document

This document is designed to assist companies wanting to create an automated electronic ordering process with their trading partners. It describes the IPO standard, developed under the direction of the Auto Care Association designed to promote electronic commerce in the automotive aftermarket.

We expect that most of you reading this document are information technology (IT) professionals charged with the task of designing and building an IPO implementation for your company. As a result this document identifies the broader technology standards and accepted best practices used when implementing web services but it doesn’t attempt to define or document them. Instead we’ll refer readers to the appropriate sources for that information.

We also expect product managers and other business analysts may review parts of this document to better understand the underlying technology standards and components that enable IPO processes.

Limitations

This IPO specification describes “public” processes conducted between trading partners, the documents they exchange and the web services framework required to exchange them. The IPO standard does not address the integration between internal systems and applications ‘behind the firewall’.

While this guide focuses on document exchanges between Buyers and Sellers without intermediaries, it certainly doesn’t preclude these solutions. Several aftermarket service providers offer IPO enabled integration solutions.

Securing web services can be relatively simple or very complex. This specification only lightly touches on basic web service security, expecting more complex security requirements will be defined by trading partners.

Development of the IPO v3 Specification

Development of this latest version of IPO started in 2010 under the IPO v2.1 Charter approved by the Auto Care Association Technology Standards Committee simultaneously with their publication of IPO v2.0. While satisfied the IPO v2 specification addressed the primary requirements of the IPO v2 Charter,
they had identified opportunities to improve Shipment and Confirmation documents that were beyond their reach in that initial release. They encouraged the committee to form a new IPO work group to address these opportunities in a subsequent IPO v2.1 release, resulting in the IPO v2.1 Charter. The IPO v2.1 Confirmation and Shipment work groups were charged with completing this work.

After extensive analysis, the IPO v2.1 Confirmation work group published a Best Practice Recommendations document in April 2012. The only structural changes to the Confirmation Base document schema was the reuse of the same document reference/identifier component used in other IPO documents.

Scope for the Shipment work group expanded beyond multiple shipments/shipping location requirements to those addressed by an EDI Advanced Ship Notice (ASN). These comparisons with EDI led the group to recognize the need to complete support for the entire order cycle by including an Invoice document in the next IPO specification. With the advent of a properly designed IPO Invoice and fully functional Shipment document comparable to an EDI ASN, the work group saw an opportunity to expand the use of IPO beyond emergency and special orders to include stock order replenishment.

In August 2011, a new Stock Orders charter replaced the previous IPO v2.1 charter, expanding scope beyond the Shipment enhancements identified in the IPO v2.1 Charter, to include an IPO Invoice document, which could serve as either functional replacements for their EDI X12 document counterparts or as near-time compliments to existing EDI batch services.

Considering the breadth and depth of change recommended by the IPO v2.1 Confirmation and Shipment work groups, and most recently the IPO Stock Orders work group, the work group and subcommittee leadership has determined the new release will be considered IPO v3.

Related documents

In addition to this document, readers may wish to consult the following:

- Auto Care Association Internet Parts Ordering Functional Implementation Guide. This document can be found in the IPO v3 documentation packet - use this link https://www.autocare.org/Technology/IPO to download the latest distribution. This guide describes the IPO exchange documents in detail and how these documents are exchanged in Buyer / Seller collaborations.

- Web services interoperability documents are located at http://www.ws-i.org/.

- Use this link to review the WS-I Basic Profile providing guidelines to secure web service interoperation: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html.
• Formatting guidelines published by the World Wide Web Consortium may be found at http://www.w3c.org

How this Document is Organized

This document is organized into three major sections; an introduction to the technical standards is first, followed by two sections offering practical guidelines - one for Buyers and another for Sellers.

1. **Technology Overview**

   The first section gives an overview of the technical standards used by IPO. This guide provides a modest overview of the technical standards required by IPO, such as HTTP and the SOAP protocol. Adopters are encouraged to consult other references for detailed information on these broader technology standards.

2. **Implementation Guidelines for Buyers**

   Since there are more Buyers than Sellers in the automotive aftermarket, we expect the largest number of implementers will be Buyers. This section provides step by step guidelines for Buyers planning to implement IPO.

3. **Implementation Guideline for Sellers**

   This section provides guidelines for Sellers supporting the IPO standard.

The appendix contains a brief overview of some general architectural patterns IPO adopters may want to consider during the design phase of their implementation. Microsoft .NET and Java J2EE platforms provide built in XML and web services capabilities that can be leveraged when implementing IPO.

Technical Background

**Standards Context**

Standards are a powerful catalyst for rapid growth for many reasons:

- **Improved interoperability**
  Trading partners exchanging information electronically using standards are less concerned about compatibility, as standards simplify integration of heterogeneous systems.

- **Higher productivity and reduced cost**
Developer tools and commercial applications that support standards reduce or eliminate the need to share a lower-level infrastructure. In areas addressed by standards, architects and developers are able to concentrate on what they need to deliver rather than how to deliver it.

- **Increased reliability**

When application behavior is standardized, it simplifies quality assurance and exception handling.

In short, standards save time and money — precious resources that may be used creating innovative and powerful applications, *rather than reinventing the wheel.*

Just as standards were instrumental in the rise of consumer-to-consumer and business-to-consumer transactions, they offer similar benefits for B2B and program-to-program trading partner transactions. IPO is built on only those standards which have emerged as the clear winners for the automotive aftermarket.

**Web Services**, as defined in the W3C Web services architecture:

[Definition: A Web service is a software system designed to support interoperable machine-to-machine interactions over a network. It has an interface described in ‘machine-readable’ format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]

A web services architecture is typically comprised of several standards, including:

- **HTTP.** As noted above, HTTP is well-known and the engine of the World-Wide Web. Its simplicity, flexibility, and ubiquity make it a logical choice for network communication.

- **SOAP.** Originally the acronym for *Simple Object Access Protocol*, now known just as SOAP, refers to an eXtensible Markup Language (XML) messaging protocol. SOAP is the means by which specific service requests are invoked, messages are passed, and results are returned.

- **Web Services Description Language (WSDL).** Another XML-based standard used to describe all aspects of a Web service - including its operations, parameters sent and results received.

Web services have become the *lingua franca*\(^1\) of service oriented architecture (SOA), with comprehensive support provided by all major infrastructure providers, including IBM, Microsoft and Oracle. Web services are used in both internal and business-to-business integration.

---

\(^1\) **lingua franca** (noun) - 1. a common language used by speakers of different languages.
Web Services Interoperability Organization (WS-I) Standards for Interoperability

Web services incorporate many different standards. In some cases it seems a new version of a standard is released before the last version has been widely adopted. Consider a moment the number of new tools employing the various versions of technology standards and the incalculable number of web services produced using those different versions. With so many variables in play, integration issues can arise even between services claiming compliance with the same standards. WS-I addresses the problem by defining guidelines to facilitate interoperability between orthogonal standards. Because IPO was designed to conform to the WS-I Basic Profile, a common framework for Web services implementation, any set of technologies conformant with this profile support IPO.

Participants Roles

Participants in an IPO transaction act in one of two basic roles: Buyer (Customer) or Seller (Supplier). A Buyer initiates a Request for Quote or Purchase Order exchange and the Seller responds with the appropriate document. The most common participants and roles are shown in Figure 1. As illustrated in the diagram, Jobbers and Warehouse Distributors may alternate between the role of Buyer and Seller depending on the role of their trading partner.

Figure 1: IPO Participants and Roles
Public & Private Processes

It’s very unlikely that two independent organizations would have the identical systems architecture. Each company’s technical infrastructure evolves from a unique combination of legacy applications, special requirements, and its own ideas about how to connect the applications it buys and builds. From hardware and operating systems to applications and processes, every business is unique. Fortunately the intrinsic design of IPO recognizes and respects this axiom.

IPO focuses exclusively on the documents and processes required for interaction between a Buyer and Seller. The IPO standard defines the format of messages to be exchanged, the sequence of those exchanges and the operations processing these messages. Buyers and Sellers are free to decide how best to integrate these public process within their own infrastructure.

Business Message Standards

IPO documents are the messages used to exchange business information between trading partners. Each IPO document uses a common architecture that can be used to connect any aftermarket company to another aftermarket company within any segment of our industry. An IPO document contains business content – such as a purchase order – independent of any communication mechanism. In theory an IPO document could be used with various types of exchange protocols but a web service implementation framework utilizing SOAP messaging is recommended.

Extensible Markup Language

Information within an IPO document is represented in eXtensible Markup Language (XML). As in most XML documents, the majority of the data exchanged is contained in elements, with the element name enclosed by a beginning tag that looks like <ElementName> and an ending tag that looks like </ElementName>. Auto Care Association has been careful to select element names, often referred to as “tag names”, which are self-describing, i.e., the list price of an item might be tagged <ListPrice> while the distributor price could be tagged <DistributorPrice>. The Auto Care Association Metadata Dictionary www.aaiaso.org contains a complete listing of all tag names used in IPO and all Auto Care Association standards, along with their definitions and document references.

You’ll notice that IPO element tag names uses medial capitals - also referred to as Pascal case, dromedary case, Upper Camel Case, etc. This convention capitalizes the initial letter of every compound word, including the first word, e.g., OrderItem, FreightTerms, BillToParty, etc. In contrast, attribute tag names use Camel Case, where the first word in a series of compound words begins with a lower case letter, such as versionId, releaseId, environment, etc.
You will also find some data elements are `nested' to form a hierarchy, so that one data element, often referred to as the parent - contains other data elements, referred to as its children. The hierarchies defined within these parent elements group related data and provide an unambiguous reference when the same child element tag name is reused in the same exchange document with a different parent element, i.e., the child element <Telephone> will contain a different value when the parent element is <ShipToParty> rather than <ShipFromParty>.

**Document Structure**

Most IPO exchange documents are divided into two primary sections; ApplicationArea and DataArea. The ApplicationArea contains information that a trading partner's infrastructure uses to initiate an IPO transaction. This includes information about the sender, the date and time of the document’s creation, and information about the intended destination or recipient.

The DataAreas of these documents inherit one of the following six base document types (XML schemas):

- RequestForQuote
- Quote
- PurchaseOrder
- Shipment
- Invoice
- Confirm

Most of these base documents are further organized into two additional hierarchies, Header and Line. The Shipment and Confirm base document types have the slightly different structure and a few infrequently used exchange documents use only the Header structure. We’ve retained the same document structure in all versions of IPO for two reasons:

1. They help organize the data into logical groups. As an example, the BillingParty component contains information about the person or company paying for an order. You'll find this component is located in the document Header structure, since there's no need to repeat this same information for each line item in the order. In contrast other components such as the BackOrderQuantity apply to a single line item, so you’d find these elements in the document’s Line component.

2. Retaining these established Header/Line structures provide backward compatibility with existing IPO implementation designs, supporting reuse of existing IPO services. While the content model
within the base documents may evolve between versions, retention of these key high-level structures make transition to latest version of IPO easier for current IPO adopters.

**From Document Exchange to Service Oriented Architecture**

Traditional Business-to-Business (B2B) standards focus on the exchange of documents between trading partners without focusing on the service that processes the document. Messages are passed from one trading partner to another with the expectation that each partner will have a gateway available to examine each incoming document and determine the type of processing required.

IPO on the other hand implements a service oriented architecture (SOA) based on Web services. SOA turns the paradigm around and makes the service and its operations the most important aspect. Hence the messages exchanged between partners become an attribute of those services.

In IPO the service used by Sellers and Buyers is implemented in collaborative exchange patterns. In the Web services model, a service may be comprised of multiple related operations. Figure 2 shows the relationship between the Web service IPOWebService and the operations this service exposes, e.g., Quote, CreatePurchaseOrder, ChangePurchaseOrder, etc.

In Figure 2, message operations are illustrated using labeled arrows. For example, the Quote Operation accepts a request message; Add RequestForQuote - and produces a response message; Add Quote.

In a Web service implementation, the Web service name (Internet Parts Order), describes the type of service offered, and the operation name (ChangePurchaseOrder for example) defines the type of functionality exposed.
Figure 2: IPO Web Service Legend

Web Services Context

Web services should be considered a family of related standards. These standards tend to build on one another and are often depicted as a stack. Figure 3 below (Source: IBM/MSFT) shows the Web services stack IPO expects.
The blue round-edge rectangles on the right hand side of the figure represent the category of service provided by the standards depicted on the left hand side of the diagram. IPO employs the Description and Messaging standards at the bottom three layers of this stack – WSDL, SOAP and XML.

The Description part of the stack is only partly defined in IPO at this time. Auto Care Association maintains sample of synchronous and asynchronous IPOv3 WSDL files at the www.aaiasoa.net/IPOv3 website. The WSDL files describe IPO services and operations for the documented collaborative exchange patterns defined in the IPO Functional Implementation Guide. These WSDL files should be downloaded by new adopters and used “as is” whenever possible, as customization of these services and their operations will compromise interoperability with potential trading partners.

While aspects of the Quality of Service and Business Process portions of the stack may be used by some advanced implementers, these are not part of the current IPO specification.

**Interaction Styles**

**Synchronous Buyer Initiated Interaction**
Some Buyers may be unable or unwilling to validate digital certificates or support a web server in order to accept unsolicited incoming web service requests. The implication here is that integration is typically Buyer driven through a synchronous request/response pattern as illustrated in Figure 4.

The Buyer implements a Web service client, depicted as the circle. The Seller implements the Web service shown as a dotted rectangle, and its operation depicted by the white rectangle. Because the generally a Buyer will expect an immediate response from the Seller after sending a special order request, i.e., the Buyer initiated a synchronous request/response interaction.

![Figure 4: Synchronous Interaction – Immediate Response](image)

There may be times where a Seller is unable to respond immediately with the expected exchange document. In that case the Seller should respond to the Buyer’s synchronous request using a Confirmation document. When a Seller responds with a Confirmation document, they are telling the Buyer they received the request but are unable to respond immediately with the expected document, i.e., their Add Quote response to the Buyer’s AddRequestForQuote. Since synchronous messaging is always limited to a single request/reply message exchange, the Buyer should periodically “poll” the Seller to retrieve the expected document, as shown in Figure 5, which illustrates this pattern in Quote collaboration.
IPO combines both interactions into a single Web service, as illustrated in Figure 6. IPO service operations support both the return of a completed functional AddQuote response or a Confirmation document confirming receipt of the request.

An order lifecycle using the synchronous request/response exchange pattern is shown in Figure 7, which combines all possible responses, both immediate and delayed. Note that each synchronous interaction is completed as a single HTTP POST session. The client (Buyer) initiates the HTTP session and is blocked until the Seller returns the response.
Figure 7: Complete Synchronous Buyer Initiated Web Services
The messages passed between Buyer and Seller operations are numbered to indicate the logical sequence. In practice these interactions may be governed by functional requirements such as an unknown product identifier or the Seller’s ability to support immediate response. Note that IPO messages are wrapped in a HTTP POST request, common for most SOAP-based web services.

**Asynchronous Interactions**

While some Buyers will be limited to IPO client implementations, others have a robust web services infrastructure that can accept Seller initiated exchange. In contrast to the synchronous exchange pattern where a Buyer always initiates an interaction, either party may initiate the exchange using an asynchronous non-blocking style of interaction. Figure 8 shows a sample interaction where the Buyer initiates a RequestForQuote operation without expecting an immediate response from the Seller. Later the Seller sends an AddQuote to the Buyer without expecting an immediate response.

![Figure 8: Async Interaction with Reliable Messaging and No Receipt Confirmation](image)

Unfortunately the HTTP transport protocol is inherently unreliable. Yet most adopters feel some measure of assured delivery is desirable in IPO interactions. In lieu of Reliable Messaging, trading partners can send an IPO Confirmation document to confirm receipt. Figure 9 shows the use of a Confirmation document in an asynchronous IPO Quote collaboration. Receipt of this Confirmation document indicates the message was received and validated but not yet processed by the trading partner system. See the Functional Implementation Guide for the specifics of Confirmation usage in this context.
This section presents alternative approaches for implementing asynchronous interactions. As already mentioned, both the Buyer and Seller must host a web service implementation to implement this style of interaction. Figure 10 shows the implementation with reliable messaging.

Note: While POST isn’t explicitly depicted in these illustrations, it’s the HTTP verb used in either single request or request/response SOAP messages. You’ll find more information on HTTP and SOAP in the SOAP Messaging Overview section.
Error! Reference source not found. shows an asynchronous implementation over HTTP with Confirm BODs providing receipt acknowledgement. Implementers of this model have a few additional responsibilities to help ensure their messages are received and processed properly:
1. Message receivers must detect and reject duplicate messages. Duplicates can be detected in at least two ways:
   a. ApplicationArea/DocumentId or
   b. DataArea/Header/DocumentIds/CustomerDocumentId or SupplierDocumentId depending on the message context.

2. Message senders will retry operations which failed due to errors that may be transient, such as:
   a. Time out without any response
   b. SOAP faults = Server
   c. SOAP faults = Client where message receiver has returned an error packaged within a SOAP fault (see section 3.2.2).

3. All implementers must maintain a log of all IPO messages to assist in reconciling processing problems.

Note: while synchronous clients are depicted as individual circles rather than long, round edge rectangles in these diagrams, this doesn’t imply these are multiple client proxies, one for each operation. This convention was used just to simplify the diagram.
Figure 11: All Async Services with Confirmation
Web Services Interoperability Organization (WS-I)

The mission of the WSI is to promote “interoperability among Web services based on common, industry accepted definitions and related XML standards support.” The organization was led by premier infrastructure vendors such as IBM and Microsoft. One of the first products of WS-I’s work is its Basic Profile 1.0, which describes how various standards should be used together to develop Web services that interoperate in real-life, practical applications. This Profile provides a proven base for successful IPO implementation.

The Basic Profile requires the use of the following standards relevant to IPO:

- Web Services Description Language (WSDL) 1.1
- Simple Object Access Protocol (SOAP) 1.1
- Extensible Markup Language (XML) 1.0 (Second Edition)
- Hypertext Transfer Protocol -- HTTP/1.1 (including HTTPS)

IPO relies on WS-I as the definitive source for Web service interoperation guidance. IPO implementers must conform to the WS-I basic profile.

SOAP Messaging Overview

SOAP is a lightweight protocol supporting application communication over the Internet and the sole message protocol recommended for IPO implementations. Applications can exchange data in SOAP messages regardless of their operating system and hardware requirements. And interoperability using SOAP doesn’t require detailed knowledge of the target application’s internals. Instead applications communicate using simple XML messages sent over HTTP.

In Buyer initiated Synchronous Buyer Initiated Interactions, SOAP request/response messages are exchanged in a single session. HTTP transport maps this message pair as a single POST operation, anticipating an immediate response. Here’s the skeleton framework of a basic SOAP Request message as shown in Example 1.

Example 1: Basic SOAP Request Message
HTTP Header

POST /ipo/IPOWebService/InternetPartsOrder.asmx HTTP/1.1
Host: dotnet.abc.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "www.aaiasoa.org/ipo/IPOWebService/InternetPartsOrder CreatePurchaseOrder"

SOAP envelope starts here...

```xml
<?xml version="1.0"?>
<soap:Envelope
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    ...
    ...
  </soap:Body>
</soap:Envelope>
```

Operation Name = CreatePurchaseOrder. Default name space is the aftermarket. BOD is ProcessPurchaseOrderBOD.

```xml
<CreatePurchaseOrder
 xmlns="http://www.aaiasoa.org/IPOv2"
>  <ProcessPurchaseOrderBOD
   xmlns="http://www.aaiasoa.org/IPOv2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.aaiasoa.org/IPOv2...
   /ProcessPurchaseOrder.xsd"

  Application and Data Area follow...

  <ApplicationArea>
    ...
  </ApplicationArea>
  <DataArea>
    <Process confirm="Always"/>
    ...
  </DataArea>
</ProcessPurchaseOrderBOD>
</CreatePurchaseOrder>
</soap:Body>
</soap:Envelope>
```
The first five lines in the Header are the HTTP POST request. The first line contains the address of the service formatted as a Universal Resource Location (URL). A required Content-Length value contains the number of characters in the body of this HTTP request, which includes everything beyond the blank line at the end of this header. The SOAPAction header line provides a ‘hint’ of the service operation being called. While SOAPAction isn’t required unlike other header information, network firewalls and other processors at the HTTP protocol-level often use this information. If it is included, the value assigned must be contained in quotes. IPO adopters should refer to the WS-I Basic profile for the specific guidance in using any SOAP Header element.

The Envelope element identifies the XML as a SOAP request and defines the namespaces being used and the Body element contains the message. The XML ‘payload’ of the message follows the requisite blank line at the end of SOAP Envelope. The XML root element identifies the IPO service operation, in this case CreatePurchaseOrder. The value of that element contains the single parameter for this operation, ProcessPurchaseOrderBOD, which ‘wrappers’ the actual document XML.

Example 2 shows a Response message. The first line shows the HTTP status code of 200 indicating a successful receipt of the SOAP Request.

Example 2: SOAP Request Response Message
Asynchronous Interactions without Reliable Messaging should return a Confirmation document, so the asynchronous Request message is the same as the synchronous example. The single HTTP POST with a confirmation response completes the exchange pattern.

Asynchronous interaction with reliable messaging is defined as a one-way web operation. Since the transport layer ensures all messages are received, there's no need for Confirmation response message.

**WSDL**

IPO services are described using the XML-based Web Services Description Language (WSDL) which is included in the IPO v2 Distribution folder.

A Web Service Description comprises the following definitions:

- **Messages**: A set of named message definitions. A message component describes the abstract format of a particular message that a Web service sends or receives.
- **Port Types**: Describes the set of messages that a service operation sends/receives.
- **Bindings**: Describes a concrete binding of a port type and associates operations to a particular message format and transmission protocol.
- **Services**: Describes the set of port types that a service provides and the ports they are provided over including the URI for the services.
- **Type definitions**: Defines the OAG BODs used as messages.

IPO adopter can start with the example WSDL from the IPO v2 Distribution, either as-is or as template for their own design. Implementers requiring a customized WSDL can distribute their version to trading IPO trading partners after modifying this generic version.

Integration Development Environment (IDE) will let you import the WSDL for a service to create either client proxy code (local constructs that make calls transparently to the service) or skeleton code for a server implementation. Even for those who are constructing their own HTTP SOAP requests, the WSDL can serve as documentation of your operations.

There will be different WSDL files depending on your interaction style as indicated in Table 1.

**Table 1 – WSDL Usage**

<table>
<thead>
<tr>
<th>Interaction Mode</th>
<th>Buyer</th>
<th>Seller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>None</td>
<td>InternetPartsOrder.wsdl</td>
</tr>
</tbody>
</table>
Async Over HTTP

<table>
<thead>
<tr>
<th>Async with messaging</th>
<th>InternetPartsOrderAsyncMessagingBuyer.wsdl</th>
<th>InternetPartsOrderAsyncMessagingSeller.wsdl</th>
</tr>
</thead>
</table>

### Transport Protocols

SOAP is not tied to any particular transport protocol. It can be used successfully with a variety of bindings including HTTP, SMTP (email), and the Java Messaging Service (JMS). That said, the only protocol discussed in the SOAP 1.1 specification is HTTP. Because HTTP is the required binding under the WS-I Basic Profile 1.0 (section 5.6.2), IPO prescribes the use of HTTP or HTTPS for secure connections.

While this section provides some high level guidance regarding the use of HTTP and SOAP status codes and faults, adopters should refer to the relevant standards documentation for an exhaustive treatment of this topic. While implementations using reliable messaging middleware such as IBM WebSphere MQ or Sonic JMS operate outside the WS-I basic profile, they are often used in controlled environments such as VPN or dedicated communication line topologies.

### HTTP Status Codes

HTTP requests result in a response that includes a status code indicating success or failure. The code consists of three digits, with the first digit specifying the general category.

- **2xx – Success.** Request received and processed OK. All successful IPO requests return an HTTP response code of 200. Implementers should examine the SOAP faults and the BODs for further information.

- **3xx – Redirection.** Resource has moved. The response content contains the new location.

- **4xx – Client Error.** Request invalid or disallowed. For example, 403 indicates “forbidden” (the server requires HTTP Basic Authentication and you either specified no username and password or they are incorrect); 404 indicates “not found” (the URL specified doesn’t specify a known resource on the server. Check that you have the correct service address).

- **5xx – Server Error.** Server problems prevent processing of the request or the web service has returned an error packaged within a SOAP fault (see 3.2.2). A response code of 500 is used for all SOAP faults (see next section for further details). If there is no SOAP message included with a
5xx response, this indicates a more basic problem with the server hosting the Web service, such as a capacity overload.

A full list of HTTP status codes may be found in section 6.1.1 of the HTTP 1.1 specification.

**SOAP Faults**

SOAP fault codes are roughly analogous to HTTP status codes. There are four broad classes of faults, indicated by strings. Specific problems within these general categories can be specified using dot notation. E.g.: Client.Authentication.

- **VersionMismatch.** The client and server are using different SOAP envelope versions, as indicated by the SOAP Envelope element's namespace. Both client and server must use http://schemas.xmlsoap.org/soap/envelope.

- **MustUnderstand.** Applies only to SOAP header interpretation problems. No SOAP headers are defined by IPO.

- **Client.** There was some problem with the request that should be corrected before resending it.

- **Server.** There was a problem at the server end. You may try sending the same request later.

See section 4.1.1 of the SOAP 1.1 specification (http://www.w3.org/TR/SOAP/#_Toc478383510) for full details on SOAP faults.

**Security**

Security considerations are an important aspect of any electronic business transaction. This broad category encompasses several related issues:

- **Authentication.** A means of identifying to one another the parties in a transaction. Often this involves providing a password.

- **Authorization.** The granting or denying of access to a service or function based on defined privileges. Authorization is normally based on authentication; i.e., once you know who the user is, you allow or deny specific requests.

- **Privacy.** Preventing transaction content from being read by anyone except the parties involved. Privacy is usually accomplished with encryption of data.

- **Data integrity.** Answers the question "Is the data that was sent the same as what was received?"
• **Non-repudiation.** The ability to prove that a transaction took place, who participated, and what the content was. Digital signatures support this element of security while also providing privacy and data integrity.

IPO allows for two main options for security:

• **None.** In certain IPO environments, it may be unnecessary to implement special security measures. For example, in transactions between an installer and a jobber, privacy concerns are minimal and the likelihood and impact of misrepresentation is minor. Considering this, and the ease of discovering and addressing such issues if they do occur, the cost and effort of implementing security may be unwarranted.

• **Secure HTTP (HTTPS) using SSL 3.0 or TLS 1.0.** This form of transport-level security included in the WS-I Basic Profile 1.0 can meet most common security requirements. For data privacy, HTTP offers 128-bit RAS encryption. With a server-side digital certificate, HTTPS can use used to authenticate the server; i.e., identify the server to the client. The reverse is also possible by installation of digital certificates on Web services clients.

Another authentication option is HTTP Basic Authentication, a simple username and password mechanism. It may be used with or without HTTPS to authenticate Web services clients.

IPO web service providers wanting to control and encrypt communications at a broader network level - as opposed to a service or connection level - may do so by implementing a Virtual Private Network (VPN). IPO Implementation using VPN is outside the scope of this document. Visit the Virtual Private Network Consortium at http://www.vpnc.org/vpn-standards.html for additional information.

In environments where requests must pass unencrypted through an intermediary, i.e., as part of the internal integration, XML-Encryption (http://www.w3.org/TR/xmlenc-core/) and/or XML-Signature (http://www.w3.org/TR/xmldsig-core) standards may be used to encrypt or sign specific document elements.

Non-repudiation may be addressed by logging each transaction, where both trading partners are responsible for maintaining logs of the messages exchanged between them. These logs can be used to track down errors in processing and resolve disputes over the disposition of orders.

**Compliance Summary**

IPO adopters must implement a minimum set of standards to claim compliance. Emerging web service standards may become mandatory in future releases of IPO. Whenever possible, these new standards will be required `in addition to' rather than `as replacements' for existing standards.
Table 2: Mandatory and Optional Standards

<table>
<thead>
<tr>
<th>Mandatory Standards</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP 1.0 or HTTP 1.1</td>
<td>Transport Level Protocol</td>
</tr>
<tr>
<td>TLS 1.0 or SSL 3.0</td>
<td>If Transport Security if required, then must be one of these.</td>
</tr>
<tr>
<td>SOAP 1.1</td>
<td>Messaging Protocol</td>
</tr>
<tr>
<td>X.509</td>
<td>If Certificates are required, we recommend this standard.</td>
</tr>
<tr>
<td>XML 1.0</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XML Schema 1.0</td>
<td>Describes and constrains XML documents</td>
</tr>
<tr>
<td>Message Log</td>
<td>Buyer and Seller must log all IPO messages sent and received.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption algorithms</td>
<td>TLS and SSL allow for the use of arbitrary encryption algorithms. No constraints on strength are imposed by IPO.</td>
</tr>
<tr>
<td>Certificate Authority</td>
<td>The choice of the Certificate Authority is a private agreement between parties</td>
</tr>
<tr>
<td>WSDL 1.1</td>
<td>Implementers may use Auto Care Association provided Web service descriptions if it is helpful.</td>
</tr>
<tr>
<td>WS-Security</td>
<td>Not yet a final standard</td>
</tr>
<tr>
<td>XML-Signature</td>
<td>Implementers can use XML-Signature as a private agreement between parties.</td>
</tr>
<tr>
<td>XML-Encryption</td>
<td>Implementers can use XML-Signature as a private agreement between parties.</td>
</tr>
<tr>
<td>WS-Reliable Messaging</td>
<td>Not yet a final standard. Implementers can choose an implementation of their choosing.</td>
</tr>
</tbody>
</table>

Implementation Guidelines for Buyers

Planning

Historically the vast majority of Buyers adopting IPO use the synchronous interaction model, although those with a robust web services infrastructure could also support asynchronous interaction. Yet many aspects of a Buyer’s implementation depend on the interaction styles supported by the Seller. So one of the
first steps in IPO adoption is to determine which exchange patterns your trading partners can support. Here’s a list of the most important information you’ll need to obtain from a potential IPO partner:

- **Supported Styles of Interaction**
  
  If the Seller supports the synchronous model, the Buyer has the option of implementing a simple IPO client. If the Seller only supports asynchronous interaction, the Buyer must implement both an IPO client to initiate requests and host a Buyer Web service to receive Seller responses.

- **Security Requirements**
  
  See 2.13 Security for information on the various security options with IPO.

- **Service Description**
  
  The most recent versions of IPO v3 WSDL files are available for review at their namespace location: www.aaiasoa.net/IPOv3/. If you’re creating an IPO web service, you can copy/paste the content in each of these files in lieu of using the WSDL files from the IPO distribution zip. You’ll need to add your web service endpoint information to a sample WSDL so your trading partner can access your service. For example, the soapbind:address element in this WSDL’s port element indicates the service uses the HTTP transport protocol and with an endpoint address of ‘http://ipo.example.com/ipo/IPOWebService/…’:

  ```xml
  <service name="InternetPartsOrderWebService">
    <port name = "InternetPartsOrderWebServiceSoap"binding="s0:InternetPartsOrderWebServiceSoap">
      <soap:address location="https://ipo.example.com/ipo/IPOWebService/..." />
    </port>
  </service>
  ```

  If you’re constructing an IPO client implementation, you’ll use a WSDL with implementation specific details provided by your trading partner; such as service location (URL, including port) and connection protocol required (HTTP or HTTPS).

**Implementing Synchronous Interaction**

There are two basic options you can use to make web service method calls from your application:

- Use a command – line tool or IDE wizard to generate native classes - referred to ‘proxies’ or ‘stubs’ - using the target WSDL as a template. These native constructs act as local intermediaries for the Web service, i.e., you call these local methods within your program and they initiate the SOAP calls for you.
• It’s possible to call to the Web service directly using your environment’s SOAP and/or HTTP tools. While a WSDL is not strictly necessary in that case, it can be a handy guide to the service’s methods and parameters shown in Table 3 below. This table lists the operations provided by the synchronous Seller web service InternetPartsOrder and the IPO documents used as input and output.

Table 3 -InternetPartsOrder Web Service

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>AddQuote</td>
<td>Successful response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>QuoteStatus</td>
<td>GetQuote</td>
<td>ShowQuote</td>
<td>Quote Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>Successful Response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>PurchaseOrderStatus</td>
<td>GetPurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>PurchaseOrder Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ChangePurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>Ack to Changed PurchaseOrder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>Confirmation</td>
<td>Received and processed as requested OR received and not yet processed. Indicated in</td>
</tr>
</tbody>
</table>
Depending on the style of communication or error condition encountered, an operation may result in the return of a Confirmation document instead of the targeted exchange document. If the Confirmation is communicating an error condition, refer to the IPO Functional Implementation Guide for information on error codes and messages. If you’re using a delayed response pattern, the Confirmation document indicates your request was received but the request has not been processed. At that point it’s up to you to poll the Seller periodically to check the status of that request. You can choose to make this polling a manual operation, or an automated one that executes at predefined intervals. Refer to the IPO Functional Implementation Guide for more information on exchange patterns.

Security

If you’re using an IPO client application, all the security requirements are established by the trading partner. There are three possible scenarios:

- **No encryption**

  It’s possible in some scenarios that no additional security measures are required. In these cases, HTTP alone is used to connect and communicate with the Seller’s service.

- **SSL 3.0 or TLS 1.0 encryption with server authentication**

  In this scenario the Buyer uses HTTPS to connect rather than HTTP. Upon connection to the Seller, you will receive the Seller’s digital certificate which you can compare to known Seller information before deciding whether to continue with a transaction.

- **SSL 3.0 or TLS 1.0 communication with client and server authentication**

  If the Seller requires Buyers to be authenticated with HTTPS and certificates, you will need to obtain and install a digital certificate. Certificates are obtained from a Certificate Authority (CA) of your choice. Versign (http://www.verisign.com) and Thawte (http://www.thawte.com/) are two established CAs whose certificates are widely trusted, but check with your Seller to determine if
they have any special certificate requirements. Some prominent Sellers may choose to act as a CA and issue their own client certificates to Buyers.

Installation details for certificates vary from operating system to operating system. In Microsoft Windows, the process is initiated by simply double-clicking the .crt file and selecting Install Certificate.

Java-based applications certificates use the keytool utility to install certificates.

**Basic Authentication**

HTTP Basic Authentication may be used in combination with any of the above security mechanisms. In this case, the Seller will give you a username and password that must be provided upon connection. Your HTTP or SOAP toolkit will provide a facility for specifying these values.

**Exception handling**

Generally any SOAP or HTTP errors encountered by Buyers can be handled by simply reissuing the request. However, the type of error received indicates whether a change in the request should be made before a retry. HTTP status codes in the 4xx range, and SOAP faults of type Client, generally indicate problems requiring modification of the SOAP request (incorrect URL, bad credentials, etc.).

Any IPO web application encountering an error that can’t be signaled to the caller via normal return exchange document constructs or Confirmation error document should throw a SOAP fault – see the Confirmation document section in the IPO Functional Implementation Guide for details. Applications calling IPO web services must check for SOAP faults. If a SOAP fault is received and the message contains a Confirmation document, it will likely have an HTTP status code of 500 and a SOAP fault type of ‘Client’. Client’s receiving an HTTP status code in the 5xx range and a SOAP fault of the type ‘Server’, generally indicate problems on the Seller’s side. If these codes are received, the Buyer should retry the same request in case there was a transient communication or access problem while attempting to connect to the Seller’s system.

**Asynchronous Buyers**

**Overview**

Buyers implementing an IPO asynchronous exchange pattern with a Seller require a web services client and server implementation. Implementation of the client portion is essentially the same as the synchronous version discussed previously in Section 3.2. The information in that section regarding synchronous security and exception handling also applies when connecting to a Seller’s asynchronous service.
Asynchronous IPO Seller services may support a standard service using HTTP and/or a service supporting reliable messaging transport protocols.

Examples of reliable messaging transports include IBM MQSeries, Microsoft MSMQ and Sun JMS, all of which ensure a message will be delivered once, and only once. It is unlikely that reliable messaging will be used over the public internet but this is a viable approach for implementations using a VPN or a private network.

Contact your Seller to determine which reliable messaging implementation is supported and be certain you use the corresponding WSDL when creating your client. Table 4 and Table 5 show each service. The difference between the two is simply in the return values: the standard service returns a Confirmation for each request, while the messaging service returns nothing.

**Table 4: Async Seller Service Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ChangePurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
</tbody>
</table>

**Table 5: Async Seller with Reliable Messaging Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>Quote request</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>PO request</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>Change request</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>Cancel request</td>
</tr>
</tbody>
</table>

The major difference between the asynchronous and synchronous implementations as a Buyer is the requirement to create and host your own internet-accessible Web Server. A Buyer web server exposes services a Seller’s system invokes to send their response to your previous request. The best way start the process of creating your own Buyer service is to add your own endpoint location information to the IPO WSDL available online at www.aaiasoa.net/IPOv3 or contained in the IPO v2 Distribution zip folder. Make sure you choose the correct WSDL depending on your decision to use reliable messaging. Table 6 and Table 7 provide a summary of the operations for each service.
Most developers use command-line framework tools or IDE wizards to automatically create a ‘skeleton framework’ for your SOAP-based web service using a WSDL file as the template. Once generated, you fill in this framework with the code required to integrate your web service with your internal applications.

**Table 6: Async Buyer Web Service Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessQuoteResponse</td>
<td>AddRequestforQuote</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ProcessPurchaseOrderCreate</td>
<td>AckPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ProcessPurchaseOrderChange</td>
<td>ProcessPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ProcessPurchaseOrderStatus</td>
<td>ShowPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ProcessShipmentStatus</td>
<td>ShowShipment</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
</tbody>
</table>

**Table 7: Async Buyer Reliable Messaging Web Service Operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessQuoteResponse</td>
<td>AddQuote</td>
<td>Quote response</td>
</tr>
<tr>
<td>ProcessPurchaseOrderCreate</td>
<td>AckPurchaseOrder</td>
<td>PO response</td>
</tr>
<tr>
<td>ProcessPurchaseOrderChange</td>
<td>AckPurchaseOrder</td>
<td>Change response</td>
</tr>
<tr>
<td>ProcessPurchaseOrderCancel</td>
<td>Confirmation</td>
<td>Cancel response</td>
</tr>
<tr>
<td>ProcessPurchaseOrderStatus</td>
<td>ShowPurchaseOrder</td>
<td>Change of PO Status</td>
</tr>
<tr>
<td>ProcessShipmentStatus</td>
<td>ShowShipment</td>
<td>Shipment Info</td>
</tr>
</tbody>
</table>

**Security**

Buyers are responsible for determining their own security requirements when hosting an IPO web service and then communicating those requirements to their trading partners. The following list provides some high-level guidance but refer to Section 2.13 for additional details.

- If you require an HTTPS connection for your service, obtain a certificate from a certificate authority (CA) and install it according to the procedure required by your Web server software.

- If your implementation requires client authentication, you’ll need to set up your Web server to support that option and inform the Seller of your requirement. If you intend to issue your own client certificate, you’ll need to create a certificate for each Seller who will be connecting and provide them with the appropriate .crt files. Certificates can be created in Linux using openssl, and Certificate Services in Microsoft Windows. Authentication of Sellers via client certificates.
should be supported by your Web server software. Configuration options include permitting access to anyone who has your certificate or limiting access to only those clients presenting specific certificate serial numbers.

If you want to use Basic Authentication to authenticate a Seller connecting to your service, create the user account(s) and assign the appropriate privileges to the web service using the utilities provided by your operating system and/or web server. Then send the username and password to the Seller using some secure method of communication.

Implementation Guide for Sellers

Overview

A first step in implementing IPO Seller services is choosing the interaction model(s) you intend to support; synchronous, asynchronous or both exchange patterns - see the 2.7 Interaction Styles for a description of these patterns. It's likely this decision will be influenced by the exchange patterns preferred by your Buyers.

If your Buyers use a simple IPO client implementation, you'll host a synchronous IPO Seller web service. Synchronous with immediate response is the most popular IPO pattern for Buyers because it's the easiest model to implement and they'll receive the information they need from the Seller’s exchange document in a matter of seconds.

Unfortunately as a Seller, the Immediate Response pattern is the most difficult implementation to support. The Seller's service must process the Buyer’s request, then retrieve all the information required to fully populate the appropriate IPO exchange document within a matter of seconds. This can be real challenge when inventory, order processing, shipping, and financial data and processing resources are located in disparate systems throughout the enterprise. While the use of any asynchronous or synchronous with delayed processing pattern would provide Sellers with additional time to collect and process all this information, those patterns are more difficult for Buyers to implement because they require a hosted IPO web service and additional processing on the Buyer side.

Security

Once you've selected a communication and exchange pattern, you and your trading partner(s) will need to determine whether a secure connection is required to exchange IPO documents, e.g., VPN connection, encryption, authentication, etc. Refer to sections 2.13 and 3.3.2 for more information on security.

WSDL Usage
You’ll use the IPO WSDL that matches the communication and exchange pattern(s) you select. Use the WSDL as a reference for developing your IPO web service. You’re free to use the programming language and application framework you choose, since these implementation details remain hidden from your trading partners using web services. If you intend to support an asynchronous interaction, make certain you select the correct WSDL based on your decision to support reliable messaging. Remember to add the location of your company’s web service in the namespace section of the WSDL. See 3.1 Planning, for a description of how to update a WSDL to reflect your Web service implementation, as those guidelines are applicable to both Buyer and Seller.

The Tables below provide a summary of the operations for each service by style of interaction.

### Table 8 - Synchronous Seller Service

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>AddQuote</td>
<td>Successful response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>QuoteStatus</td>
<td>GetQuote</td>
<td>ShowQuote</td>
<td>Quote Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>Successful Response</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>PurchaseOrderStatus</td>
<td>GetPurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>PurchaseOrder Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ChangePurchaseOrder</td>
<td>AcknowledgePurchaseOrder</td>
<td>Ack to Changed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PurchaseOrder</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Confirmation</td>
<td>Received but not processed</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>Confirmation</td>
<td>Received and processed as requested OR received and not yet processed. Indicated in OutcomeValue</td>
</tr>
</tbody>
</table>
### Table 9 - IPO Async Seller Service Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ChangePurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>Confirmation</td>
<td>Received</td>
</tr>
</tbody>
</table>

### Table 10 - IPO Async Seller Service Operations with Reliable Messaging

<table>
<thead>
<tr>
<th>Operation</th>
<th>In</th>
<th>Out</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quote</td>
<td>AddRequestforQuote</td>
<td>None</td>
<td>Quote request</td>
</tr>
<tr>
<td>CreatePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>None</td>
<td>PO request</td>
</tr>
<tr>
<td>ChangePurchaseOrder</td>
<td>ProcessPurchaseOrder</td>
<td>None</td>
<td>Change request</td>
</tr>
<tr>
<td>CancelPurchaseOrder</td>
<td>CancelPurchaseOrder</td>
<td>None</td>
<td>Cancel request</td>
</tr>
</tbody>
</table>

Most SOAP ‘aware’ Integrated Development Environments (IDE) will automatically generate a ‘skeleton framework’ for the web service based on a WSDL. Once generated, you fill in the framework with the code required to integrate this service with your internal applications.

**Publishing Your IPO Web Service**

Once your Seller service is created, generate and distribute the WSDL for your service to your Buyers, including implementation-specific details such as the location (URL, may include port) and connection protocol required (HTTP or HTTPS), see example 3 below.

**Example 3: Identifying your IPO web service**

```xml
<service name="InternetPartsOrderWebService">
    <port name="InternetPartsOrderWebServiceSoap" binding="s0:InternetPartsOrderWebServiceSoap">
        <soap:address location="https://ipo.example.com/ipo/IPOWebService/..."/>
    </port>
</service>
```
Creating the Buyer Service Client

Adoption of the asynchronous model also requires development of a web services client to interact with your Buyer’s service. Refer to Figure 9 or Figure 10 to see which exchange documents are expected as a response to service request, depending on your decision to use Reliable Messaging or an IPO Confirmation document to acknowledge receipt of a Buyer message.

There are two basic options for creating web service method calls for your application:

- Use the tools available in your development environment to generate native classes - referred to as ‘proxies’ or ‘stubs’ - using the WSDL for the Buyer service as a template. These classes act as a local intermediaries for web service operations, i.e., you call these local methods and they make the required SOAP calls for you.

- Use other SOAP/HTTP tools to make direct calls to the web service. While a WSDL may not be necessary in this case, it serves as a reference to the service’s methods and parameters.

If you’re not using a hosted WSDL at a buyer’s endpoint to generate a service proxy, you’ll need to obtain the service address, protocol and port information directly from the Buyer.

Security

If the Buyer requires you to authenticate using HTTPS, you will need to obtain and install a client digital certificate. This may be issued by the Buyer or you may be able to use one from another Certificate Administrator (CA). Installation details for certificates vary from operating system to operating system.

In Microsoft Windows, the process is initiated by simply double-clicking the .crt file and selecting Install Certificate. Java-based applications certificates use a keytool utility to install certificates.

HTTP Basic Authentication may be used in combination with any of these security mechanisms. In this case, the Buyer will give you a username and password that must be provided when connecting. Your HTTP or SOAP toolkit will provide a facility for specifying these values. Refer to other information on security in this guide, specifically sections 2.13 and 3.3.2.

Working with the IPO Distribution

Overview
This section provides an overview on working with the IPO v3 distribution, which contains this Technical Implementation Guides, Functional Implementation Guide, HTML project documentation and XML document schema.

**Getting Started**

The IPO specification is distributed in a compressed folder, containing written documentation, UML model definitions in HTML, XML schema and WSDL files. Care should be taken when moving XML schema out of these folders, as each IPO v3 exchange document is dependent on other schema contained in the BaseSchema folder. These Base Schema documents are imported/included in the IPO exchange documents using the schemaLocation attribute reference contained in each xs:import and xs:include declaration. You must update those attributes with the new location information if you decide to relocate these files. Design details for each IPO XML schema can be found in IPO Functional Implementation Guide and HTML documentation.

**Generating Code**

Most modern Interactive Development Environments (IDE) and framework tools assist developers by automatically generating C++, Java or C# implementation classes, known as proxies or stubs, from XML WSDL and schema files. In general this will include class definitions for the defined elements and complex data types and code necessary for reading the XML into the Document Object Model (DOM) and vice versa - known as serialization/deserialization. Generating this code is often as easy as opening an XML file in the IDE or invoking a command line program with the appropriate parameters – refer to Section 6 for an example.

Please note that some older and/or less expense tools may not have the ability to handle XML schema import and include references; this is required when working with IPO XML schema. IPO exchange documents use import and include declarations to reuse information contained in the “Base Schema” folder. For more information refer to the XML Schema section in the IPO Functional Implementation Guide.

**Additional Constraints**

IPO XML schema declarations are only modestly constrained in order to maximize the re-use of common definitions. The result of this approach is that very little validation can be accomplished using just the document schema.

In addition to validating the structure of an IPO exchange document using the appropriate XML schema, trading partners can further constrain the existing datatypes in exchange documents and subsequently
enforce those constraints using XSL (eXtensible Stylesheets Lanaguage) files. Constraints defined in XSL can enforce minimum occurrence, co-occurrence, and value range requirements that are difficult, if not impossible, to enforce in schemas. You and your trading partners would apply the same XSL files to a document instance either as a subsequent process to schema validation or through the use of a XSLT processor with a schema-validating parser to complete both validations in a single step.

Design Time

The XSL may be generated from a constraint language called Schematron. Constraints are first expressed using the Schematron rules, such as example 4 below.

Example 4: Schematron Script

```
<sch:pattern name="Noun Level">
   <sch:rule context="RequestForQuote">
      <sch:assert test="Header">Must have a Header component.</sch:assert>
      <sch:assert test="Line">Must have at least one Line component.</sch:assert>
   </sch:rule>
</sch:pattern>

<sch:pattern name="Header Level">
   <sch:rule context="Header">
      <sch:assert test="SupplierQuoteDocumentId">Must have DocumentIds</sch:assert>
   </sch:rule>
</sch:pattern>
```

Schematron constructs include:

- A pattern is a grouping of rules that are applied together.
- A rule is a series of assertions that will be applied to the given context.
- An assert, is a statement that test for the existence of the element and/or attribute referenced in the test attribute. If it does not exist, the message contained in the assertion is output.

Run Time

In the IPO context, it is the receiver of the document or web service provider who will be interested in checking the validity of an incoming document. Figure 12 shows the logical sequence of processing an
incoming document instance. Security and SOAP envelope/body processing is not depicted in this
diagram but would occur prior to the validation of the document itself.

![Logical Validation of IPO Documents](image)

**Figure 12: Logical Validation of IPO Documents**

### Implementation Alternatives

#### Overview

This is a brief overview of some implementation alternatives commonly considered by IPO adopters and
summary of our tooling experience.

#### Microsoft .Net Implementation

One approach to IPO adoption is to build an implementation using the Microsoft .Net Windows
Communication Framework (WCF). A WCF complaint IPO web service implementation can be built using
either a WSDL–first or Code-first approach. A WCF client implementation is developed using a Service
Reference which collects information from a metadata endpoint at the trading partner’s service.

Developers designing an IPO web service implementation using Microsoft .Net may start with command-
line utility called svcutil.exe to generate the required interfaces and implementation classes from xml
artifacts such as the IPO WSDL and schema files. Note that care must be taken to correctly reference the
path to imported and included xml schema contained in the BaseSchema folder when using this utility. As
an added bonus, Microsoft .Net 4.5 provides a new /mc switch for svcutil.exe that generates WCF
Message Contract types automatically.

#### J2EE Implementation
Java 2 Enterprise Edition (J2EE) is a widely supported alternative to Microsoft .NET, supported by prominent vendors including IBM, BEA and Oracle. These vendors provide their own Interactive Development Environment (IDE) for building J2EE applications that expose and consume web services. J2EE artifacts like Enterprise Java Beans (EJB) can easily expose web services, and descriptions of external services using WSDL can be imported by those tools to generate proxy classes.

At run time, a J2EE-compliant Application Server, such as IBM WebSphere, BEA WebLogic or Oracle 9iAS, provide services for hosting web services and invoking external web services. Figure 12, above, depicts a scenario where the Supplier has implemented their systems using a J2EE-compliant Application Server. Figure 13, below, shows a .NET Buyer and J2EE Seller.

**EAI Implementations**

Enterprise Application Integration (EAI) tools are intended to simplify integration of a diverse range of legacy and newly acquired applications. Leading vendors in the EAI Market include:

- Tibco
- WebMethods
- Vitria
- SeeBeyond

Microsoft and a number of J2EE platform Vendors including IBM, BEA, Oracle also have EAI capabilities for their platforms. EAI tools have a number of features that simplify the integration problem:

- Adapters – to connect to legacy applications
- Messaging – reliable asynchronous messaging
- Transformation – any to any mapping
- Business Process Management – advanced orchestration of complex integration processes

EAI vendors have added Web services capabilities to their platforms to better interoperate with new service-oriented architectures. IPO implementers who need to employ legacy applications and platforms in their Buyer or Supplier processes may choose to use EAI tools to “layer” these applications in order to expose and consume web services. Figure 13 shows how EAI tools can connect legacy applications like CICS/COBOL and RPG systems into a web services IPO implementation. The middleware has built in Web services capabilities in order to expose an IPO service. Behind the scenes, messaging, process
management, transformation and legacy applications adapters are employed to implement the required services.

![Figure 13: EAI Implementation](image)

Figure 13: EAI Implementation