ICE: Information and Content Exchange Protocol

Primer: Introduction and Overview

Version 2.0

2004 08 01

This version
http://www.icestandard.org/Spec/SPEC-ICE-2.0Primer.pdf
Latest version
http://www.icestandard.org/Spec/SPEC-ICE2.0d.pdf
Previous version
http://www.icestandard.org/Spec/SPEC-ICE1.1.htm

Editors:
Jay Brodsky, Tribune Media Services
Marco Carrer, Oracle Corporation
Bruce Hunt, Adobe Systems, Inc.
Dianne Kennedy, IDEAlliance
Daniel Koger, Independent Consultant
Richard Martin, Active Data Exchange
Laird Popkin, Warner Music Group
Adam Souzis, Independent Consultant

Status of this Document

This document is an approved IDEAlliance Specification. It represents a significant step towards a stable specification suitable for widespread dissemination and implementation. It has been reviewed and approved by the ICE Authoring Group of IDEAlliance.

ICE 2.0 is the first major revision of the ICE Specification. As such, ICE 2.0 is not a compatible update to the ICE 1.0 specification. This update is a response to the implementation experience that has been gained over the past four years as well as the advancement in technology and W3C Recommendations. It differs from the ICE 1.0 and ICE 1.1 specifications in that it is specifically designed to support a Web Services model for syndication, has been modularized, incorporates XML Namespaces, and moves from an XML DTD to XML Schema.

As of this publication, the ICE Specification has been organized into a set of documents. This is one document in a set of documents (ICE Primer: Introduction and Overview, ICE Cookbook, Basic ICE Specification , Full ICE Specification, ICE Schemas and Scripts, and Guidelines to Extending the ICE Protocol) intended to jointly replace ICE 1.1. It has been developed by the IDEAlliance ICE Authoring Group. New documents may be added to this set over time.

The ICE Authoring Group and IDEAlliance recommend that implementations be updated to conform to the new ICE 2.0 Specification. The new specification embraces the latest Web technologies and W3C Recommendations. It provides added functionality that greatly enhances the usability of the protocol in a very wide range of syndication applications and can provide a substantial foundation for delivering syndication solutions in a Web Services environment.

Abstract

This document describes the Information and Content Exchange protocol for use by content syndicators and their subscribers. The ICE protocol defines the roles and responsibilities of Syndicators and Subscribers, defines the format and method of content exchange, and provides support for management and control of syndication relationships. We expect ICE to be useful in automating content exchange and reuse, both in traditional publishing contexts and in business-to-business relationships where the exchange eBusiness content must be reliably automated.

Table of Contents

1. Introduction

Reusing and redistributing information and content from one Web site to another is often an ad hoc and expensive process. The expense derives from two different types of problems:

- Before successfully sharing and reusing information, both parties need a common vocabulary for content.
- Before successfully transferring any data and managing the relationship, both parties need a common messaging protocol and syndication management model.

Successful content syndication requires solving both halves of this puzzle. Fortunately, industry-specific efforts already exist for solving the vocabulary problems. Since 1998, many industries have established their own industry-specific XML vocabularies. A listing of industry XML vocabulary efforts can be found at XML.org.

ICE addresses the second problem of the redistribution and reuse of content by providing the solution for successfully transferring data and managing the syndication relationship.. Specifically, ICE enables the management and automation for the establishment of syndication relationships, data transfer, and results analysis. When combined with an industry specific vocabulary, ICE provides a complete solution for syndicating any type of information between information providers and their subscribers.

1.1 ICE Design Goals

The ICE Authoring Group defined a number of design goals for ICE based on requirements analysis and much thought and discussion.

1.1.1 ICE 1.0 Design Goals

Some of the most important design goals for ICE 1.0 are included here for reference:

NOTE: These goals are non normative. They are included here because the ICE 1.0 design goals serve as the basis for ICE 2.0 as well..

1. ICE shall be straightforwardly usable over the Internet.
2. ICE shall support a wide variety of applications and not constrain data formats.
3. ICE shall conform to a specific XML syntax.
4. The ICE requirements shall constrain the ICE process to practical and implementable mechanisms.
5. ICE shall be open for future, unknown uses.

6. Compactness of representation in ICE is of minimal importance. NOTE: this is a statement about low level encoding methodology, e.g., the use of XML in general and the particular choice of tag and attribute names in particular.
7. ICE shall keep protocol and packaging overhead to a minimum. NOTE: this is a statement about protocol overhead in the sense of round trips, complexity, and other high-level performance effects. It is not a contradiction of the previous point. The design of ICE achieves its performance objectives by optimizing the high level design of the protocol flow and state management, not by micro optimizing the spelling of individual packets.

1.1.2 ICE 2.0 Design Goals

The ICE Authoring Group extended these design goals for ICE 2.0 through a formal and open requirements process. Design goals for ICE 2.0 build on the goals for ICE 1.0. New goals for ICE 2.0 include:

1. **XML Namespaces:** The requirement is to eliminate element collisions by moving all ICE-defined elements into one or more ICE namespaces.
2. **XML Schema:** Since ICE is a protocol, it requires features such as type definitions found in XML Schemas but not supported by XML DTDs. This entails ICE DTD transforming to ICE SCHEMA but more than a straightforward translation to one that is extensible.
3. **Simplicity of Specification:** There shall be a requirement to break ICE into modules in a manner that allows for simplicity of implementation and maintains interoperability.
4. **ICE and SOAP:** ICE 2.0 needs to define the characteristics of the communication over SOAP Version 1.2**.**
5. **Express ICE as a Web service (WSDL):** There is a requirement to define the end points of the ICE conversation as WSDL, either message-oriented, RPC-oriented or both, on top of SOAP.
6. **Asynchronous Communication:** ICE must be able to support Asynchronous Communication for wireless and transient systems.
7. **ICE Subscription Management of non-ICE delivery, FTP and simple `HTTP:GET` Mechanism:** ICE 2.0 shall be able establish a subscription that may then be delivered outside the ICE protocol. E.G. use ICE subscription management to control the FTP delivery of files. ICE 2.0 is designed to handle current and future delivery vehicles, and an apparatus needs to be considered to allow for such delivery including both in-band and out-of-band delivery transport with behavior defined and in-band and out-of-band negotiation transport with behavior defined.

1.2 How ICE Relates to Other Standards

Many other standards describe how to transmit data of one form or another between systems. This section briefly discusses some of these protocols and describes their relationship to ICE.

1.2.1 XML

ICE is an application of the Extensible Mark-up Language (XML 1.1). Basic concepts in ICE are represented using the element/attribute mark-up model of XML. Note, however, that ICE is a *protocol*, not just a DTD, and so in that way differs fundamentally from other pure document applications of XML such as MathML (mathematical formula mark-up language) and SMIL (Synchronized Multimedia Interchange Language).

1.2.1 XML Namespaces

XML Namespaces 1.1 provides a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. XML Namespaces enable us to define a set of unique element names within a given context. Namespaces prevent element collisions and enable computers to unequivocally determine exact points of reference. Such unique addressing is critical to reliable messaging between Web Services. In ICE 2.0, all ICE-defined elements will be moved into one or more ICE namespaces to enable ICE to function as a Web service.

1.2.2 XML Schema

XML Schema Definition Language 1.1 is a three-part specification from the W3C that provides the capability to specify and constrain XML applications. XML Schema provides a superset of the specification capabilities of the XML DTD. Only XML Schema enables specification of type that is expected by Web services. This difference is so critical that the SOAP specification specifically states that a SOAP message “MUST NOT” contain a DTD. Since ICE is a protocol, it requires features such as type definitions found in XML Schemas but not supported by XML DTDs. ICE 1.0 was specified with an XML DTD. ICE 2.0 is specified with an XML Schema.

1.2.3 RSS

RSS is a simple mechanism for enabling the lightweight distribution of promotional links to content. RSS was designed to be simple to use and inexpensive to implement. RSS has proven quite useful for the distribution of free content, but remains limited in its ability to enforce business rules in the content syndication environment. Basic ICE is comparable in complexity and capabilities to RSS. Full ICE, on the other hand, was developed by industry content-providers and software vendors to automate the scheduled, reliable, secure redistribution of valued content for publishers and for non-commercial content providers.

1.2.4 SOAP

SOAP (Simple Object Access Protocol) 1.2 is a key enabler of Web Services through XML. SOAP enables the exchange of XML messages so that services can easily describe their capabilities and allow any other service, application or device on the Internet to easily invoke those capabilities. ICE, working with SOAP, adds the mechanisms for the management of syndication on the Web. SOAP is being widely used as transport for Web services related RPC. ICE 2.0 is designed to layer its communications on SOAP. This will enable developers and users to take advantage of their existing communication infrastructure and management services while taking advantage of ICE for their content distribution applications or content subscription activities. SOAP V1.2 became a W3C Recommendation on June 24, 2003.

1.2.5 WSDL

Web Services Definition Language (WSDL) is an XML based description language that currently describes RPC based end-points. This is currently being developed by W3C for extending RPC to enable messaging-style program end-points. ICE 1.0 has an XML-based protocol for conversation between client and server. For ICE 2.0, we are defining ICE end-points with WSDL (either message-oriented or RPC-based or both). This will eliminate the need for ICE client packages. Any WSDL to Java or any other programming language based generator will be able to generate ICE client interfaces in that programming language. This also enables customer applications to embed ICE capabilities within their applications as Web services and their Web services management infrastructure can manage their client. WSDL 1.1 is used in this document to define ICE end-points for the Full ICE Syndicator and Full ICE Subscriber. WSDL scripts are not required for Basic ICE.

1.2.6 UDDI

Internet-based Universal Description, Discovery, and Integration specification (UDDI) is a specification for distributed Web-based information registries of Web services. UDDI registries are designed to help users discover these distributed Web services. UDDI compatibility will enable subscribers to discover a server that can deliver content to their ICE client. In this sense, compatibility with UDDI can provide a discovery mechanism for the Web syndication services.

1.2.7 PRISM

PRISM is the Publishing Requirements for Industry Standard Metadata. PRISM provides an industry-standard metadata vocabulary to describe content assets. This vocabulary can work with ICE to automate content reuse and syndication processes, but it is not a syndication protocol. PRISM is a discovery mechanism and enables the selection of content that will be syndicated using ICE. There is a natural synergy between ICE and PRISM. ICE provides the protocol for syndication processes, and PRISM provides a description of the resource being syndicated. IDEAlliance hosts both working groups.

1.2.8 DOI

The Digital Object Identifier (DOI®) is a system for identifying intellectual property in the digital environment. It provides a framework for managing intellectual content, for linking customers with content suppliers, for facilitating electronic commerce, and enabling automated copyright management for all types of media. DOI does not address the management of content syndication, rather it provides a unique identifier, that when used with a syndication messaging and management protocol (ICE) will enable content management and distribution. ICE 2.0 will enable the use of DOI as a unique content identifier.

1.2.9 XrML

XrML (Extensible Rights Markup Language) is an XML vocabulary that provides a universal method for securely specifying and managing rights and conditions associated with all kinds of resources including digital content as well as services.. XrML compatibility with ICE is important for specifying and managing rights during the process of syndication. ICE 2.0 is designed such that it is compatible with XrML.

1.2.10 CDF

Channel Definition Format (CDF) specifies the operation of push channels. Like ICE, it defines a mechanism for scheduling delivery of encapsulated content. ICE builds on some of the concepts of CDF, such as delivery schedules. Note that ICE goes well beyond what CDF can do; CDF has no notion of explicit subscription relationship management, asset management, reliable sequenced package delivery, asset repair operations, constraints, etc.

We expect ICE will be useful for server-to-server syndication to distribute and/or aggregate content to/from various push servers, whereas CDF is useful for server to browser applications.

1.2.11 OSD

The Open Software Description (OSD) Format automates distribution of software packages. OSD focuses on concepts such as package dependencies, OS requirements, environmental requirements (such as: how much disk space does a software package require), etc. ICE has very little overlap or relationship to OSD.

We expect ICE to be useful for server to server syndication to distribute and/or aggregate content to/from one OSD server to another, whereas OSD continues to be useful for its intended domain of distributing and installing software directly to target desktop and work group server machines.

1.2.12 P3P

Quoting from [P3P-arch]: *The Platform for Privacy Preferences (P3P) protocol addresses the twin goals of meeting the data privacy expectations of consumers on the Web while assuring that the medium remains available and productive for electronic commerce*. When ICE is being used to share user profile information from one business to another, it is the responsibility of the applications on both sides of such a relationship to enforce the appropriate privacy policies in accord with the principles described in P3P, as well as in accord with any governing laws. ICE is merely the transport mechanism for those profiles and is not involved in the enforcement of user profile privacy principles.

1.2.13 WebDAV

Quoting from [WebDAV]: *WebDAV (Distributed Authoring and Versioning) specifies a set of methods, headers, and content types ancillary to HTTP/1.1 for the management of resource properties, creation and management of resource collections, name space manipulation, and resource locking (collision avoidance).*

WebDAV addresses a collaborative authoring environment and has very little overlap with ICE.

1.2.14 HTTP DRP

Quoting from [NOTE-DRP]: *The HTTP Distribution and Replication protocol was designed to efficiently replicate a hierarchical set of files to a large number of clients. No assumption is made about the content or type of the files; they are simply files in some hierarchical organization.*

DRP focuses on the differential update of information organized as a hierarchy of files. As such, it could be used to solve a portion of the data transfer problems addressed by ICE, but only for those content syndication situations that are file centric. ICE solves a more general problem of asset exchange, where assets may not necessarily be files in a hierarchy. ICE also addresses explicit subscription relationship management, asset management, reliable sequenced package delivery, asset repair operations, constraints, etc. whereas DRP addresses none of those.

1.2.14 Atom

Atom defines a feed format for representing and a protocol for editing Web resources such as Weblogs, online journals, Wikis, and similar content. The feed format enables provision of a channel of information by representing multiple resources in a single document. The editing protocol enables agents to interact with resources by nominating a way of using existing Web standards in a pattern. ICE is a general syndication protocol that can apply to any type of content, and provides for subscription management and confirmation of delivery. ICE does not address reading and editing of resources.

1.3 Definitions

1.3.1 Requirement Wording Note

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

In the HTML version of this specification, those key words are **CAPITALIZED BOLD**. Capitalization is significant; uncapitalized uses of the key words are intended to be interpreted in their normal, informal, English language way. Bold face is not significant, and is used to aid comprehension, but the bold font is non normative and the absence of a bold font **MUST NOT** be given any semantic interpretation.

1.3.2 ICE Semantic Definitions

These definitions are used throughout this document. Readers will most likely not fully understand these definitions without also reading through the specification.

catalog

A package of subscription offers. A Subscriber pulls a catalog package (by convention Syndicators will offer the catalog as a subscription with subscription-id=”1”) from a Syndicator, and uses the offers within the catalog to initiate the ICE subscription protocol.

collection

The result of a Subscriber processing all package deliveries in a single subscription, that is, the current content of a subscription. This is equivalent to the set of all items that a Syndicator would deliver in a full update of a subscription.

This is not necessarily every item a Syndicator would transmit over time in a given subscription, because of incremental update.

full update

A set of all items within a subscription are delivered with each update. Basic ICE only allows for this update method.

ICE

Information and Content Exchange.

incremental update

A set of only changed items within a subscription are delivered with each update. Basic ICE does not allow for this update method.

ICE/HTTP

The specific binding of the ICE protocol to the HTTP protocol.

ICE/SOAP

The specific binding of the ICE protocol to the SOAP protocol.

item

A single delivery instance of an arbitrary data type. For example, if a database record were being distributed, each field might be encapsulated as an item. Or, if a prospectus consisting of an HTML file and two GIF image files is being distributed, each of the files would be an item (within an item group).

item group

A delivery instance of one or more items. For example, if a prospectus consisting of an HTML file and two GIF image files is being distributed, each of the files would be an item within a single item group.

message

The abstract concept of an atomic unit of communication. In this specification, the term *message* does not denote any specific protocol structure; rather, it is used to denote an abstract communication concept.

offer

An abstract representation of content that can be subscribed to along with delivery policies.

package

A single delivery instance of a group of items. For example, a package is a single issue of a parts manual or a single set of headlines. A package is the atomic unit of information distribution in ICE. A package is also used to distribute ICE offers.

package sequence

An ordered series of packages delivered over time.

Receiver

Generic term referring to the target of an ICE request. The term Receiver is used when it is possible for either the Subscriber or the Syndicator to be the party receiving the request.

Request

A message asking for the performance of an operation. Requests in ICE are messages carried by the SOAP payload.

Requester

Generic term referring to the initiator of an ICE request.

Responder

Generic term referring to the recipient of an ICE request.

Response

A message containing the results of an operation. Responses in ICE are messages carried by SOAP payloads.

Sender

Generic term referring to the originator of an ICE message. The term Sender is used when it is possible for either the Subscriber or the Syndicator to be the party sending the message

Subscriber

One of the two parties in an ICE relationship (the other one being the Syndicator). The Subscriber uses ICE to obtain information and content from the Syndicator.

subscription

An agreement to deliver a package sequence from a Syndicator to a Subscriber. There may be many independent subscriptions between a Syndicator and a Subscriber.

subscription element

A persistent identifier of all versions of an item or item group in a subscription. The subscription element may have many versions over time, and thus may have been represented by different items. For example, a company logo is a single subscription element that can be updated over time. Every subscription element has a unique subscription element ID assigned by the syndicator.

subscription offer

A proposed set of parameters for a particular subscription. Within ICE, the term *subscription offer* has a precise meaning directly related to the corresponding protocol data structure; do not confuse the usage of the term "offer" in this specification with the more generic and abstract concept of offers in the business world sense.

Syndicator

One of the two parties in an ICE relationship (the other one being the Subscriber). The Syndicator uses ICE to send information and content to the Subscriber.

1.4 Technical Decisions

The Authoring Group went through several major topics of discussion while designing ICE, and some of the decisions reached are of sufficient interest to warrant recording the thought processes that led to them.

1.4.1 ICE 2.0 Constraints

During the development of ICE 2.0, the ICE Authoring Group once more searched for an existing schema and constraint definition language for the content carried by the ICE payload that would meet the ICE requirements. For example, XML schema can be used to specify the `dateTime` format.

With the writing of ICE 2.0, the ICE Authoring Group feels that:

- Constraints are a necessary part of a syndication solution.
- W3C's XML Schema provides a workable solution to defining constraints for the specification
- In addition, ICE 2.0 provides specific error and status codes for handling constraint violation errors.

With ICE 2.0, XML Schema is used to specify and manage ICE constraints for packaging and messaging. ICE 2.0 does not specify any particular constrain language for the content that ICE carries. Further, the ICE Authoring Group now considers the definition of such a content constraint language out of scope for ICE. Today, other specifications, such as PRISM and XrML, own the domain of specifying constraints and/or metadata for content carried by ICE.

Note that a conforming ICE implementation need not implement any constraint processing at all such as XML Schema validation. Constraint processing for ICE is entirely a quality of implementation issue. Its presence or absence has no effect whatsoever on the interoperability of two ICE implementations, because nothing in the protocol state machine flow depends on constraint processing.

1.4.2 Defining ICE 2.0 using an XML-Schema

ICE 1.0 used DTD syntax to define the format of the ICE protocol. While an XML DTD was used to define the format for ICE 1.0, XML Schema has been selected for the definition format of ICE 2.0. This selection was made so that ICE could work over SOAP and function as a Web service. SOAP uses XML Schema for its definition and specifically disallows specification by XML DTDs for interoperability with SOAP.

It is important to note that XML Schema is used as the definition format for ICE, but that validation against the schema is not strictly required. In fact there are two places where XML Schema validation is implied by ICE 2.0:

- A Receiver **MAY** perform validation on incoming ICE messages.
- A Sender **MUST** send only valid ICE messages.

Note, however, that "validation" could in principle be implemented in a variety of ways. A Receiver **MAY** use any alternate representation of ICE syntax, and perform some alternate form of validation against that representation, as long as the results are AS-IF the governing ICE XML Schema had been used.

1.4.3 Use of SOAP Transport Mechanism

Because one of the design goals of ICE 2.0 is to enable ICE as a Web service, the capability of ICE to function over SOAP is critical. ICE 2.0 will remain a transport independent protocol. However this ICE 2.0 Specification will explicitly discuss binding of the generic ICE protocol over the SOAP transport mechanism and term that ICE/SOAP.

For ease of implementation, Basic ICE is restricted to the SOAP Response Message Exchange Pattern (`HTTP:GET`) as a transport mechanism. In this SOAP pattern an `HTTP:GET` retrieves whatever data is identified by a URI, so where the URI refers to a data-producing process, or a script which can be run by such a process, it is this data which will be returned, and not the source text of the script or process.

Full ICE adds the SOAP Request-Response Message Exchange Pattern. This implies that both the Syndicator and Subscriber have request and response capabilities.

1.4.5 Security

The ICE protocol (ICE 1.0 and ICE 2.0) deliberately does not address security, because the required levels of security can be achieved via existing and emerging Internet/Web security mechanisms.

In the specific case of digital signatures, non repudiation, and similar concepts, two things have happened that have steered the Authoring Group away from the notion of having digital signatures inside ICE itself:

- Separate efforts are underway to define digital signing standards for XML documents. The ICE Authoring Group felt that duplicating such work within ICE was not warranted.
- Defining digital signing standards for XML documents is quite tricky, and requires defining a canonical text representation of the documents (because the digital hash functions hash the *textual* representation of a document, not its *logical* representation). The ICE Authoring Group did not want to define its own, possibly conflicting, canonical representation rules to solve this problem.

Independent of any future XML digital signing standards, ICE implementations can achieve necessary security using a variety of methods, including:

- Encryption can be accomplished at the transport level, e.g., via SSL, PGP, or S/MIME.
- Applications can agree to send digitally signed content as items within the ICE protocol, with verification performed at the application level (above ICE).
- Syndicators and Subscribers can be authenticated using certificates implemented at the transport level.
- Syndicators can offer extended ICE subscriptions where the specific content structures to be encrypted as well as the encryption types may be negotiated using subscription extension described in ***Error! Reference source not found.***

Also, for interoperability, Syndicators and Subscribers need to agree on how they will negotiate the security parameters for a given relationship. This may be done inside of ICE by using protocol extension. Or it may be done outside of ICE by, for example, an agreement to use SSL at a certain level of encryption, or by some other external means.

1.4.6 Internationalization Issues

Few internationalization issues occur at the protocol level at which ICE operates, but four specific issues are worthy of NOTE:

1. **Support for International Character Sets**. ICE relies on capabilities in XML for encoding and supporting international character sets.
2. **Other Protocol Text Strings.** The ICE protocol sometimes uses string values as semantic identifiers. For example, a `<sender name=` encodes the sender's name as a textual string. These textual strings are intended as arbitrary tokens representing a specific concept; they are not intended for presentation and thus have no impact on internationalization issues.
3. **Language identifier for textual data**. Some ICE elements are specifically designed for the transport of textual data intended for use by humans (defined as textType). For example, text is expected in the `<icesub:text>` element of `<icesub:business-term` element or in the `<icedel:description` of an item. ICE provides a `xml:lang` attribute in all places where human readable text is being transported and might require an identification of its specific language encoding. When used, the `xml:lang` attribute **MUST** be filled in according to standards RFC-1766 (Tags for the Identification of Languages) and ISO-639 (Code for the representation of names of languages) as is required by the XML Specification.

1.4.7 ICE Modularity

One of the early design goals for ICE 2.0 was the requirement to provide modularity for ICE. Modularity will, in effect, enable users of the ICE specification to select certain modules for implementation and leave others unimplemented. Modularity will also enable ICE to interoperate with other specifications, such as SOAP and Web services specifications, in a seamless fashion ICE modularity is documented in more detail in *3.1 ICE Modularity*.

1.4.8 ICE Simple Datatypes

In order to specify simple datatypes used in ICE 2.0, a simple datatypes schema was developed for inclusion in each of the ICE schema definitions. This is discussed in more detail in

2.6 ICE Simple Datatypes and is found in its entirety in *ICE: Scripts and Schemas.*

1.4.9 ICE Namespaces

In order to enable interoperability in the Web services environment, ICE 2.0 uses unique namespaces to support ICE messaging, ICE subscription and ICE delivery. This is discussed in more detail in *2.7 ICE Namespaces*.

1.4.10 ICE XSD's

ICE XML Schemas can be found on the ICE Standard website.
`http://www.icestandard.org/Spec/V20/schema/icesimpledatatypes20040801.xsd`
`http://www.icestandard.org/Spec/V20/schema/ice-message20040801d`
`http://www.icestandard.org/Spec/V20/schema/ice-subscribe20040801d`
`http://www.icestandard.org/Spec/V20/schema/ice-delivery20040801d`

1.4.10 ICE WSDL Scripts

In addition to the schema definitions for ICE 2.0, sample WSDL1.2 scripts are provided to help define ICE as a Web service. The ICE 2.0 Specification provides the following WSDL scripts that define operations and bindings for Full ICE:
`http://www.icestandard.org/Spec/V20/wsdl/ice-subscriber-full20040801.wsdl`

`http://www.icestandard.org/Spec/v20/wsdl/ice-syndicator-full20040801.wsdl`

The WSDL scripts provided in this specification are non-normative but are provided for reference. These scripts are WSDL 1.2 compliant. WSDL 2.0 is currently a Working Draft of the W3C. As such, the WSDL scripts provided within this specification may need to be updated when WSDL 2.0 becomes a W3C Recommendation.

Note: WSDL scripts are not required for Basic ICE, which does not require a Web Services architecture.

1.5 Structure of this Document

The remainder of this document is organized as follows:

- Chapter 2 provides an overview of the ICE protocol. In this chapter the basic roles of the syndicator/subscriber and request/response are discussed. The use of XML Namespaces, and overview of the XML schemas, the XML syntax for ICE elements and attributes, the types of identifiers and status codes are discussed.
- Chapter 3 introduces ICE levels of capability. In this section the modules of the ICE specification are introduced. The ICE features that must be supported by a

Basic ICE implementation, a Full ICE implementation, and ICE extension mechanisms are presented.

1.6 ICE Roadmap

As of this publication, the ICE Specification has been organized into a set of documents. This ICE Primer: Introduction and Overview is one document in a set of documents that make up ICE 2.0. Other documents in the set include:
(,, Full ICE Specification, ICE Schemas and Scripts, and Guidelines to Extending the ICE Protocol) intended to jointly replace ICE 1.1.

- *ICE 2.0: Primer* provides an introduction and overview of ICE 2.0. It describes the design rationale and defines the levels of an ICE implementation.
- *ICE 2.0: Cookbook* provides developers with implementation recipes for increasing levels of ICE functionality.
- *ICE 2.0: Basic ICE Specification* describes a basic ICE implementation. This section provides a detailed description of basic protocol operations.
- *ICE 2.0: Full ICE Specification* describes a full ICE implementation. This section provides a detailed description of complete protocol operations.
- *ICE 2.0: ICE Schemas and Scripts* contains the four ICE XML schemas and two sample WSDL Scripts
- *ICE 2.0: Guidelines to Extending the ICE Protocol* describes how to extend the ICE protocol. This section provides details about how XML Namespaces can be used to extend the ICE protocol.

1.7 Conventions

This document contains a number of constructs that are identified:
- Examples (XML instance files)
- XML schema fragments
- Figures

In addition, as specific ICE tags are documented, they will be set in a typewriter face with an open bracket, but no closing bracket. Namespace designations will be used. For example `<icesub:get-package` would be used when discussing the get package element.

2. ICE Overview

Two entities are involved in forming a business relationship where ICE is used. The *Syndicator* produces content that is consumed by *Subscribers*. The Syndicator produces a subscription offer from input from various departments in an organization. Decisions are made about how to make these goods available to prospects. The subscription offer includes terms such as delivery policy, usage reporting, presentation constraints, etc. An organization's sales team engages prospects and reaches a business agreement typically involving legal or contract departments. Once the legal and contractual discussions are concluded, the technical team is provided with the subscription offer details and information regarding the Subscriber. The subscription offer is expressed in terms that a web application can manage (this could be database records, an XML file, a plain text file, and so on). In addition, the technical team may have to set up an account for the subscriber entity, so that the website can identify who it is accessing the syndication application.

The Subscriber receives the information regarding their account, their subscriber identification and the syndicator endpoint. At this point, actual ICE operations can begin. The important point to understand is that ICE starts after the two parties have already agreed to have a relationship, and have already worked out the contractual, monetary, and business implications of that relationship.

The ICE protocol covers three general types of operations:

- Messaging
- Delivery / Transport / Packaging
- Subscription

From the ICE perspective, a relationship between a Syndicator and a Subscriber starts off with some form of subscription establishment. In ICE, the Subscriber typically begins by obtaining *offers* from the Syndicator. The Subscriber then *subscribes* to particular offers with specified delivery transports, protocols and schedules and the Syndicator acknowledges the *subscription*.

The relationship then moves on to the steady state, where the primary message exchanges center on transport, messages and data delivery. ICE uses a *package* concept as a container mechanism for generic data items. ICE defines a *sequenced package model* allowing Syndicators to support both incremental and full update models. Basic ICE is limited to the full update model. Full ICE implementations must support either update model. ICE also defines push and pull data transfer models as well as out-of-band transfer.

Managing exceptional conditions and being able to diagnose problems is an important part of syndication management; accordingly, ICE defines a mechanism by which *faults* can be exchanged in a standardized manner between (consenting) Subscribers and Syndicators.

Finally, ICE provides a number of mechanisms for supporting miscellaneous operations, such as the ability the ability to query and ascertain the status of the subscription.

2.1 Simple ICE Scenarios

Two simple scenarios are used throughout this specification as the source for examples: syndication of news headlines from an online publisher to other online services, and syndication of a parts catalog from a manufacturer to its distributors. 2.1.1 Headline Scenario
An online content provider, Headlines.com, allows other online sites to subscribe to their headline service. Headlines.com updates headlines three times a day during weekdays, and once each on Saturday and Sunday. A headline consists of four fields: the headline text, a small thumbnail GIF image, a date, and a URL link that points back to the main story on Headlines.com.

Subscribers who sign up for the headline service can collect these headlines and use them on their own site. They display the headlines on their own site, with the URL links pointing back to Headlines.com.

For an extra fee, subscribers may harvest the actual story bodies from Headlines.com and thus incorporate content directly into their own site instead of linking back to Headlines.com.

2.1.2 Parts Scenario

A jet powered pencil sharpener manufacturer, JetSharp.com, wants to keep its distributors up to date with the latest parts and optional accessories catalog at all times. It is very important to JetSharp that its distributors always have easy access to the latest service bulletins, and also that they have the latest information about optional accessories and the corresponding price lists.

Each item in the JetSharp parts catalog consists of some structured data, such as price, shipping weight, and size, and also contains unstructured data consisting of a set of HTML files and GIF images describing the product.

The JetSharp catalog is huge, but, fortunately, changes fairly slowly over time.

2.2 Protocol Overview

The ICE protocol is primarily a request/response protocol that allows for fully symmetric implementations, where both the Syndicator and Subscriber can initiate requests. This fully symmetric implementation is known as *Full ICE*. The ICE protocol also allows for a *Basic ICE* implementation where only the Subscriber can initiate requests (i.e., no agent that would be considered a "server" resides on the Subscriber machine).

There are several key concepts that form the foundation of the ICE protocol.

2.2.1 Messages, Requests and Responses

ICE uses *message* exchange as its fundamental protocol model, where a *message* is defined for the purposes of this specification to be a SOAP payload as specified by the ICE 2.0 Specification.).

ICE messages contain *header* information along with *requests* and *responses*. A *request* asks for the performance of an operation. For example, when a Subscriber wishes to initiate a relationship by obtaining a catalog of offers from a Syndicator, the Subscriber sends the Syndicator a message containing a `<get-package` request with a subscription id equal to “1” where “1” is by default the request for the Syndicator’s package of offers. Similarly, in this case the response contains the results of the operation and returns a package of offers.

2.2.2 Request/Response model

Every logical operation in ICE is described by a request/response pair. All operations are forced to fit this model; thus, a valid ICE protocol session always comprises an even number of messages when it is in the idle state (i.e., there is a matching response for every request).

2.2.3 Subscriber/Syndicator, Requester/Responder, Sender/Receiver

The Subscriber and Syndicator assume several different roles during ICE protocol operations: Subscriber versus Syndicator, Requester versus Responder, and Sender versus Receiver.

The definition of Subscriber and Syndicator is based on the business relationships: the Syndicator distributes content to the Subscriber. These terms are capitalized throughout this specification wherever they refer specifically to the roles of the parties in an ICE relationship, as opposed to the general concepts of subscribing and syndicating.

The definition of Requester/Responder is based on who initiates the ICE operation. The initiator is the *Requester*, and the other party, who performs the operation, is the *Responder*. It is possible for a Syndicator to be either a Requester or a Responder, depending on the particular operation. The same is true for a Subscriber. For example, when a Subscriber initiates a `<icedel:get-package` request to a Syndicator, the Subscriber is the Requester. When a Syndicator pushes a `<icedel:package` request to a Subscriber, the Syndicator becomes the Requester and waits for a `<icedel:package-confirmation response from the Subscriber, who in this instance is also the Responder.`

Finally, the concept of Sender and Receiver are used in this specification to describe the relationship with respect to the transmission of a single message. A message travels from *Sender* to *Receiver* (and this thus forms the definition of Sender and Receiver).

Note that an ICE operation inherently consists of a Request/Response pair. Thus, the Requester starts out being a Sender, sending a message, containing a request, to the Receiver. The request could be a SOAP message or a simple `HTTP:GET`. The Receiver of this first message becomes the Responder. When the Responder has performed the operation and wishes to return the results, the Responder becomes the Sender of a message containing the response, and the initial Requester is now the Receiver.

2.3 Bindings of ICE

Because one of the design goals of ICE 2.0 is to enable ICE as a Web service, the capability of ICE to function over SOAP is critical. While ICE 2.0 will remain a transport independent protocol, thus the ICE 2.0 Specification will explicitly discuss binding of the generic ICE protocol over the SOAP transport mechanism and term that ICE/SOAP. In addition to the specification of ICE 2.0 with an explicit binding to SOAP, ease of implementation also dictates that ICE 2.0 also enable the use of `HTTP:GET` ICE/HTTP as a transport mechanism. The bindings for ICE 2.0 are spelled out in the WSDL scripts.

2.3.1 Binding ICE to SOAP Response Message (`HTTP:GET`) Pattern

In Basic ICE all messages are initiated with a SOAP Response Message Pattern (`HTTP:GET`). The `HTTP:GET` retrieves whatever data is identified by a URL. The body of the response will be an XML message with a SOAP envelope and ICE messages as defined by the ICE 2.0 Specification.

2.3.2 Mapping ICE to SOAP

For Full ICE, an explicit binding to SOAP is provided. This binding can be found in the following (partial) WSDL script:

```
<binding name="ice-syndicator-full-binding" type="tns:ice-
syndicator-full-portType"/>
```

ICE 2.0 was specifically designed to function as a Web service and to take advantage of SOAP as a messaging protocol. The ICE message header was designed to be carried in the SOAP header and the ICE fault, delivery and subscription mechanisms were designed to be enclosed in the SOAP body.

ICE uses XML as the format for its message header, delivery and subscription elements. All ICE message elements **MUST** be formatted in accordance with the XML 1.1 specification. Furthermore, ICE message elements **MUST** be well formed and **MUST** be valid according to the ICE XML Schema Definitions.

This document does not repeat the general rules for proper XML encoding; readers are expected to refer to the XML specification.

To understand how ICE works with SOAP, see Figure 2.1.

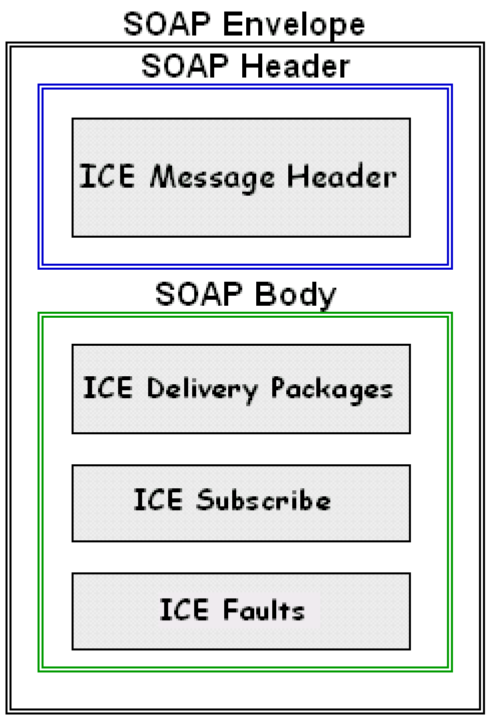

Figure 2.1. ICE carried by SOAP

For either Full ICE or Basic ICE, content is encoded in an ICE/SOAP format. This is simply an XML file where ICE messages are wrapped in a SOAP envelope..

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/12/soap-
envelope'>
  <env:Header>
<icemes:Header
xmlns:icemes='http://icestandard.org/ICE/Spec/V20/message'
timestamp="2003-03-03" message-id="m0056">
  <icemes:Sender name="mycompany"
role="http://icestandard.org/ice/2.0/role/syndicator"
   sender-id="http://www.xxyz.org"/>
  </ice:Header>
  </env:Header>
  <env:Body>
<icedel:package
xmlns:icemes='http://icestandard.org/ICE/Spec/V20/delivery'
 new-state="P3" old-state="P2"
  fullupdate="false"  package-id="012" subscription-id="3">
   <icedel:add subscription-element="offer3">
 <icedel:item-refurl="http://mysite.com/text.htm"/>
</icedel:add>
</icedel:package>
  </env:Body><env:Envelope>
```

2.4 ICE Syntax and Format

ICE 2.0 uses XML as the format for all ICE messages. XML schemas are used to define simple datatypes, the ICE message header and status codes and ICE delivery and subscription elements.
ICE makes extensive use of XML attributes for representing values. The following requirements apply to the interpretation of attribute values:

- Unless explicitly indicated otherwise, leading and trailing white space characters in attribute values **MUST** be ignored. For example, the following two attribute values are equivalent:

```
"equivalent"
" equivalent "
```

- All attribute values must conform to simple datatyping rules as expressed in the ICE Simple Datatypes Schema.

2.5 Identifiers

ICE defines a number of identifiers that control the access to content and enable content management throughout the syndication process.

2.5.1 Subscriber and Syndicator Identifiers

ICE uses globally unique identifiers for identifying Subscribers and Syndicators. The globally unique identifier for the Subscriber and Syndicator should conform to the *Universal Unique Identifier* defined by the Open Group [OG-UUID]. Note that if a given installation sometimes functions as a Subscriber and sometimes functions as a Syndicator then it **MAY** use the same UUID as its identification in both roles. Although not recommended, an ICE implementation may use a unique identifier not based on the open group standard, such as email addresses or domain names. Once the identifier has been generated, it must be treated as opaque by all parties.

The UUID format as specified consists of 32 hexadecimal digits, with optional embedded hyphen characters. Per the requirements in the *Universal Unique Identifier* specification, ICE implementations using the UUID **MUST** ignore all hyphens when comparing UUID values for equality, regardless of where the hyphens occur. Also, note that comparisons **MUST** be case insensitive.

2.5.2 Other Identifiers

As distinct from the Subscriber UUID and the Syndicator UUID as outlined by the Open Group, ICE does not define the format of other identifiers it specifies except for uniqueness constraints. All other identifiers function as being unique only within a certain scope. For example, a subscription identifier is generated by a Syndicator when the relationship between a Subscriber and a Syndicator is first established. The identification string used for the subscription ID need only be unique within the domain of all subscription identifiers generated by that Syndicator for the Subscriber.

The following table describes each identifier in ICE, its scope, a description of where in an ICE message the ID value is assigned, the role of the party that assigns the ID value, where this ID value is referenced, and finally, the section in the specification where the identifier is discussed.

Identifier	**Scope**	**Where assigned**	**Assigned by**	**ID Referenced by**
Syndicator's Unique Identifier	Unique identifier of a Syndicator	When ICE syndicator created	Entity wishing to use ICE to syndicate content.	sender-id attribute on `<sender` element or receiver-id attribute on `<receiver` element (depending on role)
Subscriber's Unique Identifier	Unique identifier of a Subscriber	When ICE subscriber created	Entity wishing to use ICE to subscribe to content	sender-id attribute on `<sender` element or receiver-id attribute on `<receiver` element (depending on role)
Message ID	Unique across all messages from a sender to a receiver	message-id attribute on message <header element	Sender assigns	message-id attribute on message `<header` element
Offer ID	Unique within the catalog of offers made by a Syndicator to a Subscriber.	offer-id attribute on <offer element	Syndicator assigns.	offer-id attribute on `<offer` element
Package ID	Unique across all packages within a subscription	package-id attribute on <package element	Syndicator assigns	package-id attribute on `<package` and `<confirmation` elements
Subscription ID	Unique across subscriptions from a Syndicator to a Subscriber	subscription-id attribute on <subscription element	Syndicator assigns	subscription-id attribute on <cancel, <get-package, <get-status, <confirmation, <cancellation, <subscription, <offer, <package, elements
Subscription Element ID	Unique within a subscription	subscription-element-id attribute on <group, <add, <remove-item element	Syndicator assigns	subscription-element-id attribute on `<group, <add, <remove-item` element

Many attributes in ICE contain as values the identifiers described in the previous table and use them to track and signal specific states in the syndication relationship.

2.6 ICE Simple Datatypes

One important reason for migrating from the ICE 1.0 XML DTD to the ICE 2.0 XML Data Schema is to benefit from the ability to specify simple datatypes that XSD offers. ICE simple datatypes are defined in an ICE simple datatypes schema. Datatypes for elements and attributes within ICE are specified here. The following shows an example of the datatype definitions for “dateTime” and “time”. Note that each simple datatype is documented to explain the intended usage.

```
<xs:simpleType name = "dateTime">
 <xs:annotation>
  <xs:documentation>the pattern here expresses the
restriction that datetimes in ICE must be in the UTC time
zone</xs:documentation>
</xs:annotation>
  <xs:restriction base = "dateTime">
<xs:pattern value = ".*Z"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name = "time">
  <xs:annotation>
<xs:documentation>the pattern here expresses the
restriction that times in ICE must be in the UTC time
zone</xs:documentation>
  </xs:annotation>
  <xs:restriction base = "time">
<xs:pattern value = ".*Z"/>
  </xs:restriction>
</xs:simpleType>
```

This document does not repeat the general rules for XML datatyping; readers are expected to refer to the XSD specification to understand the datatypes utilized by ICE.

2.7 ICE Namespaces

XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. XML Namespaces enable us to define a set of unique element names within a given context. Namespaces prevent element collisions and enable computers to unequivocally determine exact points of reference. Such unique addressing is critical to reliable messaging between Web services. In ICE 2.0, all ICE-defined elements are moved into one of three ICE namespaces to enable ICE to function as a Web service and utilize SOAP messaging.
The ICE 2.0 namespaces are:

- xmlns:icemes = “ http://icestandard.org/ICE/Spec/V20/message”
- xmlns:icedel = “http://icestandard.org/ICE/V20/delivery”
- xmlns:icesub = “http://icestandard.org/ICE/V20/subscribe”
- xmlns:icesdt = “http://icestandard.org/ICE/V20/simpledatatypes”

NOTE: The namespace definition is not a URL that can be directly resolved. Rather it is simple an identifier for the namespace. The prefixes used in this specification are examples. Anyone is free to assign their own namespace prefixes. Implementers should honor the namespace declarations rather than matching the prefix strings used here.

2.8 ICE Message XSD

The ICE message schema is defined within ice-message.xsd as the "`icemes`" namespace. This schema defines structures relating to the ICE message itself. This includes message header information and ICE status codes. In ICE 1.0, before SOAP, ICE had its own envelope and the ICE message header fell within the ICE envelope. Today, however, ICE uses the SOAP envelope and SOAP header. The ICE message is carried within the SOAP header.
The ICE message schema contains header information that is specific to syndication. In addition, it contains definitions for `<icemsg:ping`, `<icemsg:ok` and `<icemsg:status-code` diagnostic messages. See Figure 2.2.

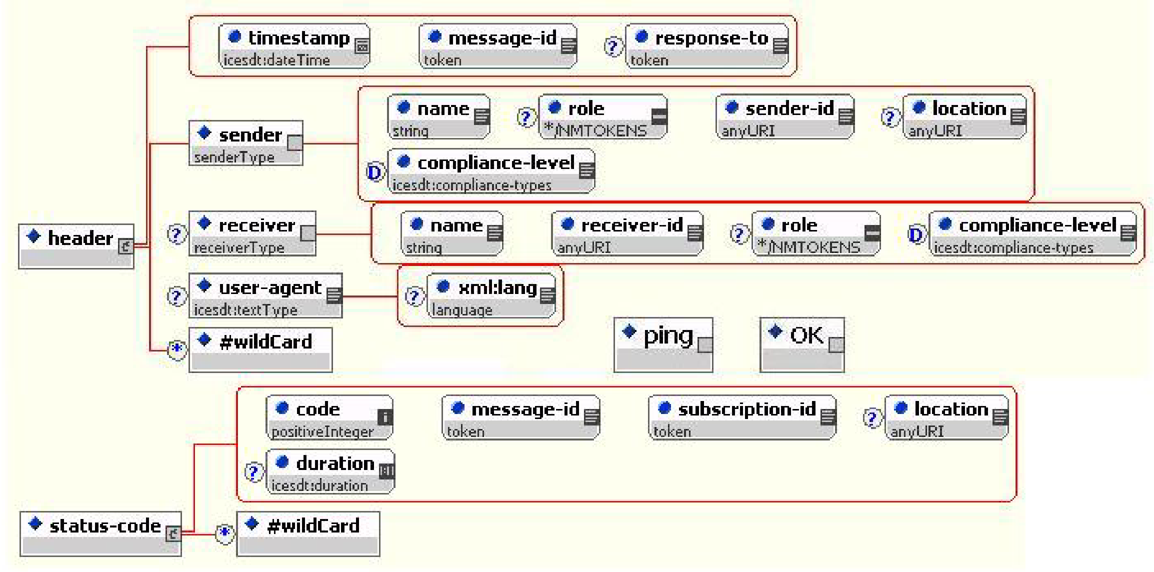

Figure 2.2. ICE message XSD

The ICE message uses the simple datatypes defined within the simple datatype module. For example, the timestamp uses the "`dateTime`" datatype that was discussed previously.

Note: The ICE message schema also contains definitions for ICE status codes. See *2.12 ICE Status Codes*.

2.9 ICE Delivery XSD

ICE delivery is defined in a schema module with the `icedel:` namespace. This module defines the elements that support the delivery of syndicated content and is carried within the SOAP body. See Figure 2.3.

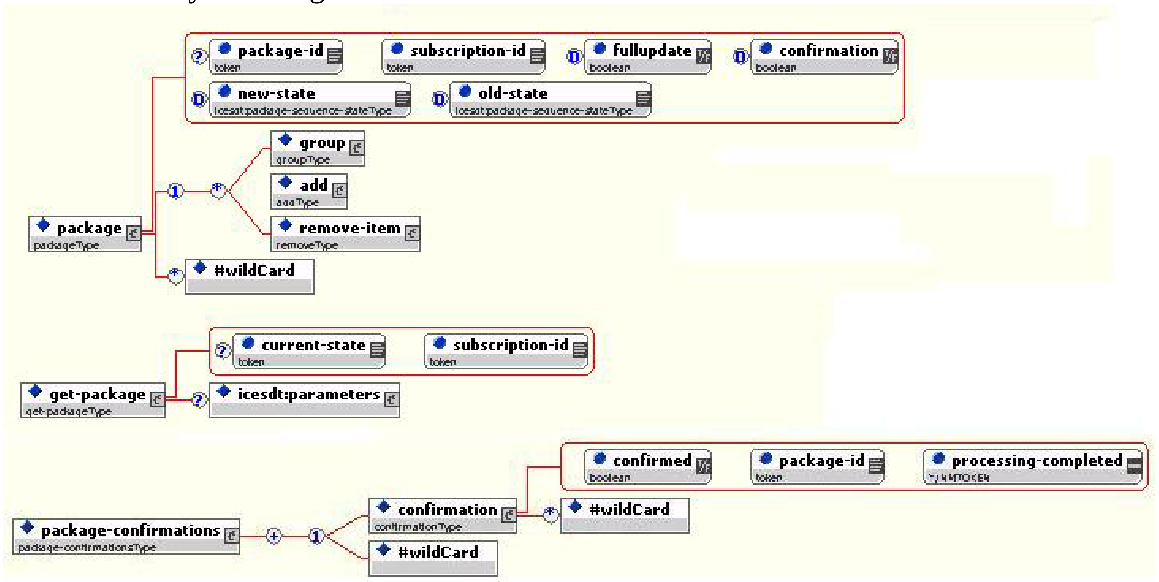

Figure 2.3. ICE delivery elements

ICE delivery is most often made up of packages. Packages may directly contain content from another XML namespace, indicated by `#wildcard` or `<icedel:packages`. Two kinds of ICE packages include those bearing or point to content and those containing a catalog of subscription offers. ICE delivery also provides for the `<icedel:get-packages` request and a `<icedel:package-confirmations` function.

2.10 ICE Subscribe XSD

The ICE subscription module is used to establish and cancel subscriptions for syndicated content. It is defined within the `icesub:` namespace. See Figure 2.4.

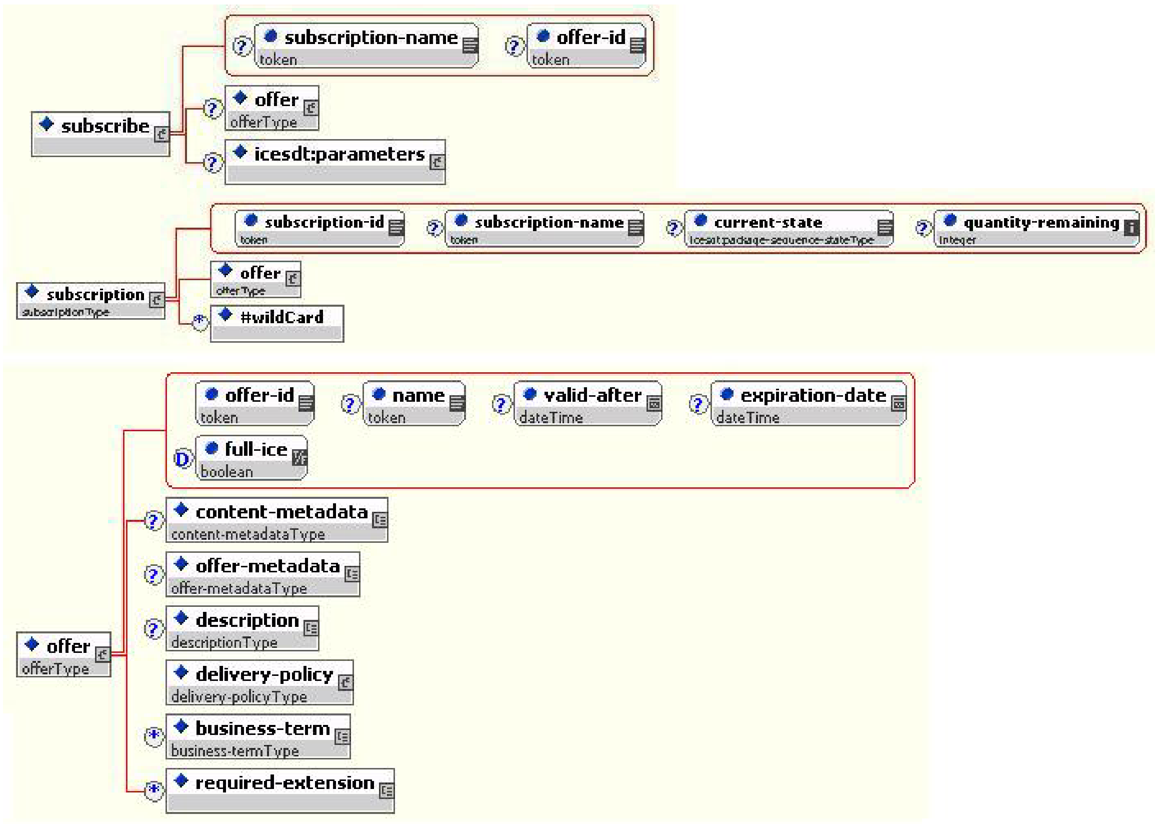

Figure 2.4. ICE subscription elements

Each ICE subscription contains one offer that will be subscribed to. Attributes on the offer identify it uniquely. Each ICE subscription offer must contain a delivery policy. The delivery rule within the delivery policy defines how and when content will be delivered. See Figure 2.5.

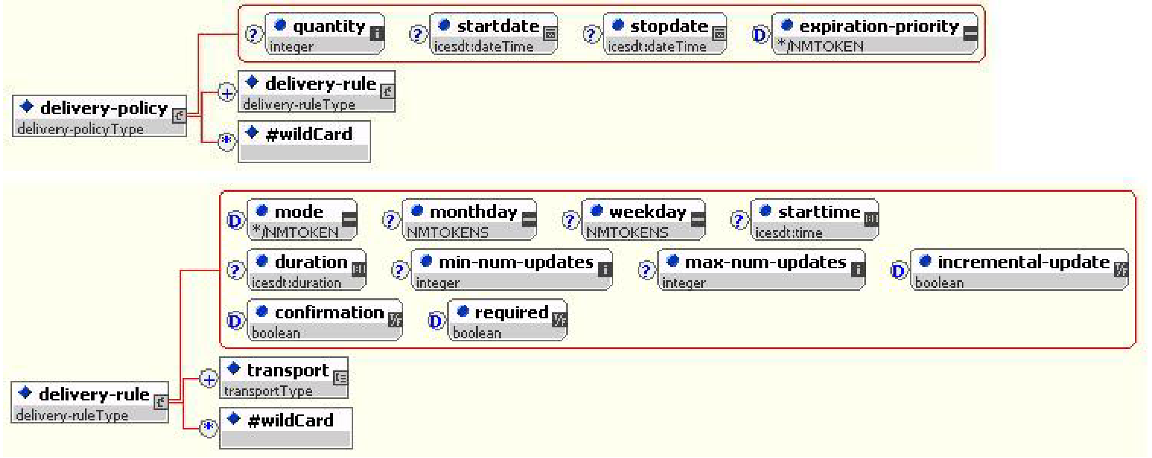

Figure 2.5. ICE Delivery Policy and Delivery Rule Structures

In addition, the ICE subscription allows for the subscriber to cancel a subscription, for the syndicator to verify cancellation and for the subscriber to get the status of a subscription. See Figure 2.6.

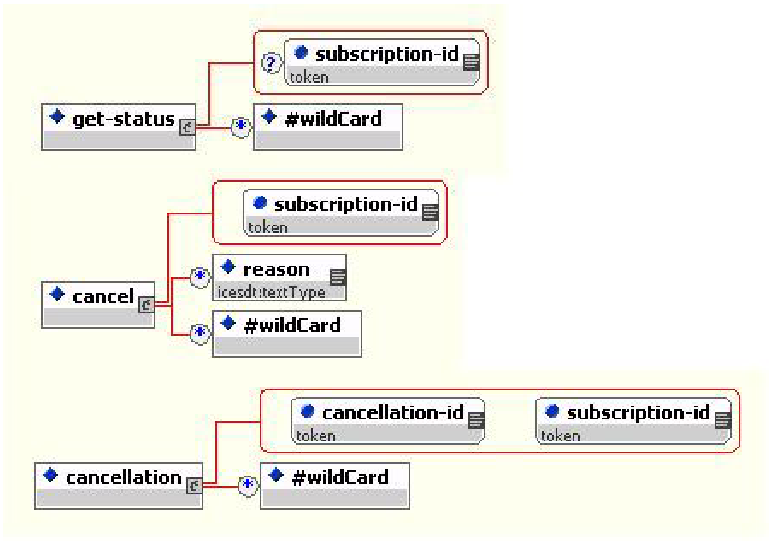

Figure 2.6. ICE Subscription Management Elements

2.11 ICE Message Header

The `<icemes:header` contains header information that fits inside the SOAP header and specifies information specific to ICE syndication messages. See Figure 2.7.

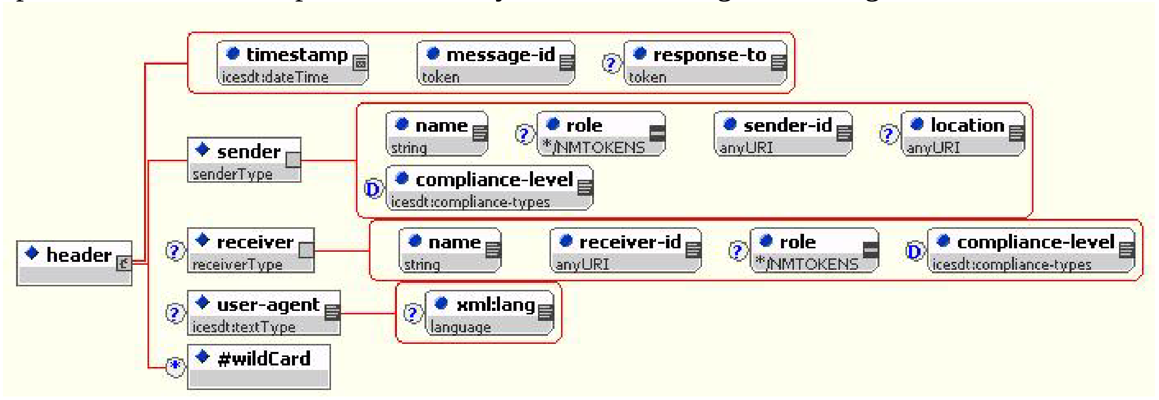

Figure 2.7 ICE Message Header Structure

2.11.1 Message Header Attributes

The `<icemes:header` has 3 attributes that are used to identify the message.

- **timestamp**
  Required. Indicates the date and time the message was sent.
- **message-id**
  Required. A unique identifier across all messages between a sender/receiver that identifies the message.
- **response-to**
  Optional. This attribute is an echo of a message to which this message is responding. It is the previous message-id.

2.11.2 Message Header Elements

The is made up of a required `<icemes:sender` element and optional `<icemes:receiver` and `<icemes:user-agent` elements.

2.11.2.1 Sender

The `<icemes:sender` provides information about the sender of this message. The element is empty and has 5 attributes:

- **name**
  Required. This attribute is a string that is used to indicate the sender name.
- **role**
  Optional. This attribute provides a mechanism to indicate the role of the sender. Values include "`syndicator`" and "`subscriber`"
- **sender-id**
  Required. The unique identifier of the sender.
- **location**
  Optional. This attribute is a URI that points to the origin (sender) of the message.
- **compliance-level**
  Default. This attribute indicates the ICE compliance level of the sender. The values are "`basic`" and "`full`". The default is set to "`basic`".

2.11.2.2 Receiver

The `<icemes:sender` provides information about the sender of this message. The element is empty and has 4 attributes:

- **name**
  Required. This attribute is a string that is used to indicate the sender name.
- **role**
  Optional. This attribute provides a mechanism to indicate the role of the sender. Values include "syndicator" and "subscriber"
- **receiver-id**
  Required. The unique identifier of the sender.
- **compliance-level**
  Default. This attribute indicates the ICE compliance level of the receiver. The values are "`basic`" and "`full`". The default is set to "`basic`".

2.11.2.3 User-Agent

The `<icemes:user-agent` element provides a text field to describe a user-agent, if one is employed. If present, the `<icemes:user-agent` gives the software program used by the original client. This is for statistical purposes and the tracing of protocol violations. It should be included.

The `<icemes:user-agent` text field has a very specific format as defined by the W3C HTTP Protocol. The first white space delimited word must be the software product name, with an optional slash and version designator. Other products, which form part of the user-agent, may be put as separate words.

```
  <icemes:user-agent>  LII-Cello/1.0  libwww/2.5
  </icemes:user-agent>
```

2.12 ICE Status Codes

ICE uses the familiar Internet protocol paradigm of three digit status values in responses to protocol operations. This paradigm was chosen because it is well understood and is suited to both machine-to-machine communication and human interpretation.

2.12.1 Relationship Between SOAP Faults and ICE Status Codes

The ICE status codes are carried within `<icemes:status-code`. ICE status codes travel within the SOAP Body / SOAP Fault. See the following example:

```
<?xml version='1.0' ?>
<env:Envelope
 xmlns:env=”http://www.w3.org/2002/12/soap-envelope”
 xmlns:rpc=”http://www.w3.org/2002/12/soap-rpc”>
  <env:Header>
<ice:Header timestamp=”2003-03-03” message-id=”m0056”>
  <ice:Sender name=”mycompany”
 role=”http://icestandard.org/ice/2.0/role/syndicator”
   sender-id=”http://www.xxyz.org”/>
  </ice:Header>
  </env:Header>
  <env:Body>
   <env:Fault>
 <env:Code>
   <env:Value>env:Reciever</env:Value>
   <env:Subcode>
   <env:Value>ice:202</env:Value>
   </env:Subcode>
 </env:Code>
 <env:Reason>
<env:Text xml:lang="en-US">Package sequence state already
current</env:Text>
 </env:Reason>
 <env:Detail>
<ice:status-code code=”202”
 reason=”Package sequence state already
 current” subscription-id=”xxx”/>
 </env:Detail>
   </env:Fault>
 </env:Body>
```

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

2.12.2 ICE Status Code Format

The format of an ICE status code is described by the following schema fragment in ICE 2.0:

```
<element name = "status-code">
  <complexType>
   <attribute name = "code" use = "required" type =
"xs:positiveInteger"/>
   <attribute name = "message-id" use = "required" type =
"xs:token"/>
   <attribute name = "subscription-id" use = "required"
type = "xs:token"/>
   <attribute name = "location" type = "xs:anyURI"/>
   <attribute name = "duration" type = "icesdt:duration"/>
   <anyAttribute namespace = "##other" processContents =
"lax"/>
  </complexType>
</element>
```

The attributes on `<icemes:status-code` are:

- **code**
  Required. Three digit status/error code, as explained further below.
- **subscription-id**
  Required. The `subscription-id` of the message being referenced by this code.
- **message-id**
  Required. The `message-id` of the request referenced by this code, or, in some cases, the `response-id` of the response referenced by this code.
- **location**
  Optional. The location is a URL returned along with `code=431` (Failure fetching external data) to indicate a new fetch location.
- **duration**
  Optional. The duration is returned along with `code=422` (Schedule violation, try again later) to indicate the duration of the wait before trying again.

2.12.3 Defined Status Codes

The defined status codes are shown below. Each bullet item contains the three-digit code `positiveInteger` value, the corresponding phrase, and a description in italics. Note that the phrase and the description in italics is part of the explanation and not part of the status message.

When generating codes:

- Senders **MUST** supply a three digit `code=` value from the set defined here.

When receiving codes:

- Receivers **MUST** understand all the three digit codes described in this specification.
- Receivers **MAY** treat unrecognized codes not defined in the ICE specification, or 9xx codes, in an implementation specific manner. As a quality of implementation issue, receivers could implement user interfaces allowing customized handling or mapping of unknown codes to specific actions; however, this specification does not require them to do so.

The status values defined by ICE are:

2xx: Success

- 200 OK
  The operation completed successfully.
- 201 Confirmed
  The operation is confirmed. This code is returned when requesting confirmation of package delivery.
- 202 Package sequence state already current
  A Subscriber requested a package update, but the Subscriber is already in the current package sequence state, i.e., there are no updates at the moment.

3xx: SOAP level Status Codes

These indicate something about the SOAP message itself, as opposed to the individual requests and responses within the SOAP message. These codes have one very explicit and important semantic: they are used when the SOAP message could not be properly interpreted, meaning that even if there were multiple requests in the SOAP message, there will be only one code in the response. For example, if the SOAP message had been corrupted, it might be so corrupted that it isn't even possible to determine how many requests it contains, let alone respond to them individually.

The specific codes are:

- 320 Incompatible version
  *The ICE protocol version used in the request is not supported. NOTE: The ICE protocol versions are transmitted as part of the message header, implementations may look there to decide what appropriate corrective actions to take. Implementations **must** follow the version rules. This could also be generated for incompatibilities between conformance levels.*

4xx: Request level Status Codes

These indicate errors caused by an inability to carry out an individual request. Note that in some cases there are similar errors between the 3xx and 4xx class; the difference is whether or not the error is supplied as a single, message level error code (3xx) or whether it is supplied as a per request code.

- 400 Generic request error
  Generic status code indicating inability to comprehend the request. Usually, it is better to send a more specific code if possible.
- 401 Incomplete/cannot parse
  The request sent is severely garbled and cannot be parsed. Note that in most cases, a message level error (301) might be more appropriate.
- 402 Not well formed XML
  The request sent is recognizable as XML, but is not well formed per the definition of XML. This is available as both a message level error and as a request level (4xx) error. Whether a given implementation attempts to interpret not well formed XML so as to generate request level (4xx) errors versus. Message level (3xx) errors is a quality of implementation issue.
- 403 Validation failure
  *The request failed validation according to the Schema. This is available as both a message level error and as a request level (4xx) error. Whether a given implementation attempts to interpret not well formed XML so as to generate request level (4xx) errors versus. Message level (3xx) errors is a quality of implementation issue. Note that Receivers **SHOULD** perform validation on incoming ICE messages, but are not required to. Senders **MUST** send only valid ICE messages or they are in error; however, the ability to detect invalid messages is a quality-of-implementation issue for the Receiver, and Senders **MUST NOT** assume the Receiver will perform an XML validation on their messages.*
- 404 This error intentionally left blank
- 405 Unrecognized sender
- 406 Unrecognized subscription
- 407 Unrecognized operation
- 408 Unrecognized operation arguments
- 409 Not available under this subscription
  The Requester has referenced something not covered by the subscription referenced in the request.
- 410 Not found
  Generic error for being unable to find something, for example a subscription that has expired.
- 411 Unrecognized package sequence state
  The package sequence identifier supplied by the Sender is not understood by the Receiver.
- 412 Unauthorized
- 413 Forbidden
- 414 Business term violation
- 420 Constraint failure
  *Compliant implementations **MUST NOT** send this message if the constraint was not specified in the subscription.*

- 422 Schedule violation, try again later.
  The request was made at an incorrect time. For example, trying to get a package update outside of the agreed upon timing window.
- 430 Not confirmed
  Generic error indicating the operation is not confirmed.
- 431 Failure fetching external data
  The receiver could not follow an external reference (URL) (`<icedel:item-ref`*) given to it by the sender. Note that in ICE 2 only the Subscriber is permitted to reply with this code. A Syndicator* ***MUST NOT*** *reply with this code.*
- 440 Sorry
  Used by the Syndicator to reject a subscribe request.

5xx: Implementation errors and operational failures

These indicate errors caused by internal or operational problems, rather than by incorrect requests. Note that, like all other codes except for the 3xx series, 5xx codesmust be sent individually with each response; if the error condition or operational problem prevents the Responder from resolving the original message down to the request level, use a 3xx code instead.

- 500 Generic internal responder error
  Catch-all for general problems; recovery/retry behavior unspecified.
- 501 Temporary responder problem
  Too busy, update in progress etc. Eventually an identical retry request might succeed.
- 503 Not implemented
  The server does not implement the requested operation.

6xx: Pending State

These codes indicate a state condition where the Subscriber is expected to send something to the Syndicator, or vice versa.

- 602 Excessive confirmations outstanding
  The Syndicator had requested confirmation of package delivery, and now refuses to perform any additional operations until the Subscriber supplies the confirmations (positive or negative).

7xx: Local Use Codes

These codes are reserved for use by the local ICE implementation and **MUST NOT** ever be sent to another ice processor over the transport medium. The intent is that this range of codes can be used by the local ICE implementation software to communicate transport level error conditions, or other specific local conditions, using the `ice-code` mechanism in a way guaranteed to not collide with any other usage of `ice-code` values.

9xx: Experimental Codes

ICE implementations **MUST NOT** use any codes not listed in this specification, unless those codes are in the 9xx range. The 9xx range allows implementations to experiment with new codes and new facilities without fear of collision with future versions of ICE.

How a given system treats any 9xx code is a quality of implementation issue.

3. ICE Conformance Levels

ICE 2.0 defines three levels of conformance. These levels of conformance spell out the features of ICE that must be supported for that level of conformance. The definition of levels of conformance enables software vendors to develop ICE applications that are interoperable.

- Basic ICE software can be expected to interoperate with other software that supports Basic ICE.
- Full ICE software can be expected to interoperate with other software that supports Full ICE.
- Full ICE software can be expected to interoperate with other software that supports Basic ICE.

ICE features can be viewed as modules of the specification. In this chapter we will examine these modules and define which features make up each level of ICE conformance.

NOTE: The ICE 2.0 Specification has purposely limited its scope to define Basic ICE and Full ICE. Advanced syndication operations are allowed for within the specification as Optional ICE extensions. The idea here is to allow for implementations to extend the ICE protocol in such a way that advanced syndication operations may be allowed for in a predictable and controlled manner.

3.1 ICE Modularity

One of the early design goals for ICE 2.0 was the requirement to provide modularity for ICE. Modularity, in effect, enables users of the ICE specification to select certain modules for implementation and leave others unimplemented. Modularity enables ICE to interoperate with other specifications, such as RDF or PRISM, in a seamless fashion. The ICE modules correspond to features of ICE for each level of conformance. See Figure 3.1.

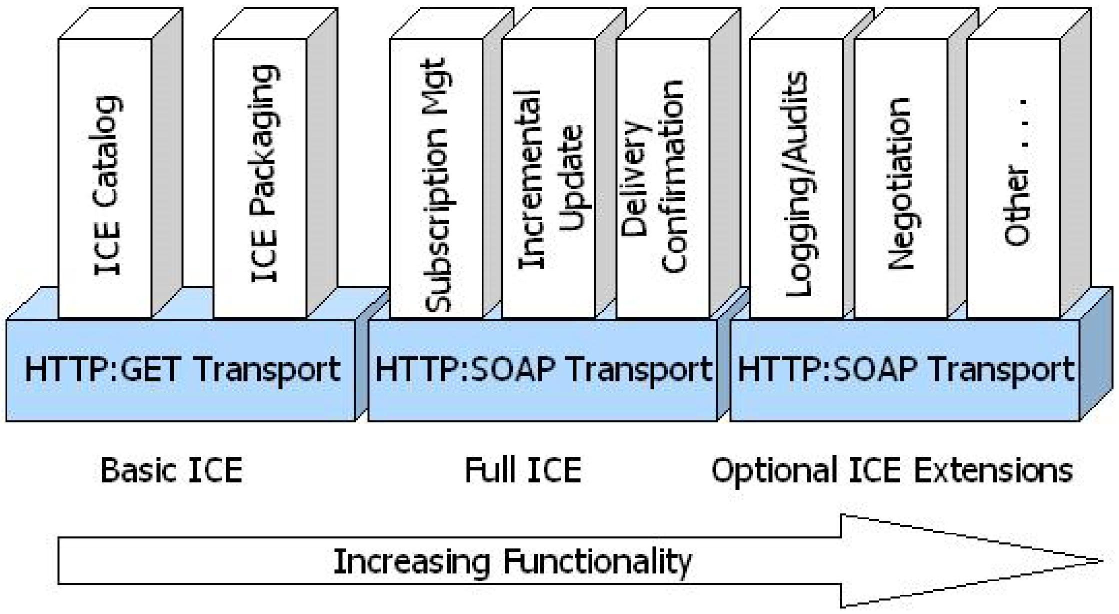

Figure 3.1. ICE modules by increasing functionality

3.1.1 SOAP Response Message Exchange Pattern (`HTTP:GET`) Transport

Basic ICE, the simplest form of ICE, relys on the SOAP Response Message Pattern (`HTTP:GET`) transport, also known as REST (REpresentational State Transfer) . The binding for this transport does not need to be defined by a WSDL script. . This replaces the concept of the “minimal subscriber” that was part of ICE 1.0.

3.1.2 Message / Package Delivery

Package delivery is required for every level of ICE since that is the essence of syndicating content. Package delivery as a feature includes the following:

ICE message header

ICE package and get-package

3.1.3 SOAP Transport

Because one of the design goals of ICE 2.0 is to enable ICE as a Web service, the capability of ICE to function over SOAP is required. ICE 2.0 remains a transport independent protocol. However Full ICE explicitly requires the binding of the generic ICE protocol over the SOAP transport mechanism using the SOAP Request-Response Message Pattern. The binding for this transport is defined in the ice-syndicator-full WSDL script.

```
  <!--SOAP Binding  -->
  <binding name="ice-syndicator-full-binding"
type="tns:ice-syndicator-full-portType">
<soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>

<!-- OPERATIONS GO HERE -->

  </binding>
```

3.1.4 Subscription Management

Full ICE adds the capability to establish subscriptions with delivery policies as well as to check the status of the subscription and cancel the subscription. While subscription management is central to ICE-based syndication, a simple form of ICE, Basic ICE, enables syndication without the added sophistication of subscription management.

3.1.5 Incremental Updates

ICE 2.0 allows for two kinds of updates for subscription content. The simplest update is the full update mechanism. A full update is a complete replacement of all the content of a subscription. Full ICE allows for incremental updates in which only the content that is new or changed is replaced. See *ICE 2.0: Full ICE Specification.*

3.1.6 Delivery Confirmation

The ICE 2.0 Specification provides messages by which a Subscriber can provide confirmation that a package was received and processed. See *ICE 2.0: Full ICE Specification.*

3.1.7 Logging

The ICE 2.0 Specification defines ICE logging capabilities from ICE 1.x as an optional extension. This extension will allow a Syndicator to request the protocol event logs of the Subscriber, and vice versa, as an aid for debugging and diagnosis. To learn how to extend ICE 2.0 to include optional logging functions, see *ICE 2.0: Extending the ICE Protocol.*

3.1.8 Negotiation

The ICE 2.0 Specification defines parameter negotiation as an optional extension. This extension provides a means for Syndicator and the Subscriber to reach mutually agreeable subscription operation. The model also permitted a Syndicator or Subscriber to define and negotiate other parameters of importance to both the subscription, and the Syndicator/Subscriber relationship and permitted semantic extension through generalized parameter negotiation. To learn how to extend ICE 2.0 to include optional parameter negotiation functions, see *ICE 2.0: Extending the ICE Protocol*.

3.2 More about Modularity

ICE modules correspond to features of ICE for each level of conformance:

- The Full ICE Syndicator WSDL describes the set of operations that implementers will have to support in order to satisfy Full ICE conformance. The ICE Subscriber WSDL describes the set of operations that implementers will have to support on the subscriber side to satisfy Full ICE conformance.
- Neither the Basic ICE Syndicator nor the Basic ICE Subscriber have an end point, hence there is no requirement for a WSDL script. Basic ICE utilizes the simple `HTTP:GET` mechanism only.

3.3 ICE Levels of Conformance

ICE 2.0 defines three levels of conformance. These levels of conformance spell out the features of ICE that must be supported for that level of conformance. The definition of levels of conformance enables software vendors to develop ICE applications that are interoperable. So Basic ICE software can be expected to interoperate with other software that supports Basic ICE. And Full ICE software can be expected to interoperate with other software that supports Full ICE.

3.3.1 Basic ICE

The Basic ICE level of conformance provides for very simple syndication functionality. In fact, all that Basic ICE enables is for the Syndicator to post messages to a URL where the Subscriber can "get" them:

- `<icedel:package` (this is limited to a single package)
- `<icemsg:status-code`

In Basic ICE, the Subscriber initiates all messages with `HTTP GET` to URLs on the Syndicator. Basic ICE does not allow for subscription management capabilities. The

Syndicator sends no messages to the Subscriber in Basic ICE. Basic ICE has no requirement for either the Syndicator or the Subscriber to establish a "listener" for push messages. Refer to *ICE 2.0: Basic ICE Specification* for Basic ICE features/modules.

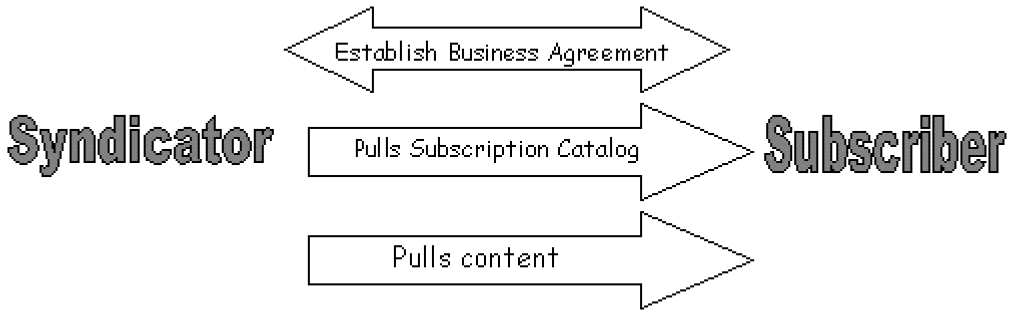

Figure 3.1 Basic ICE capabilities

3.3.2 Full ICE

A Full ICE implementation implements all the features of the ICE 2.0 specification. Full ICE implementations must support SOAP transport bindings and adds messages to support subscription management. Additional messages include:

- `<icesub:subscribe`
- `<icesub:subscription`
- `<icesub:cancel`
- `<icesub:cancellation`
- `<icesub:get-status`
- `<icesub:status`
- `<icedel:package-confirmations`

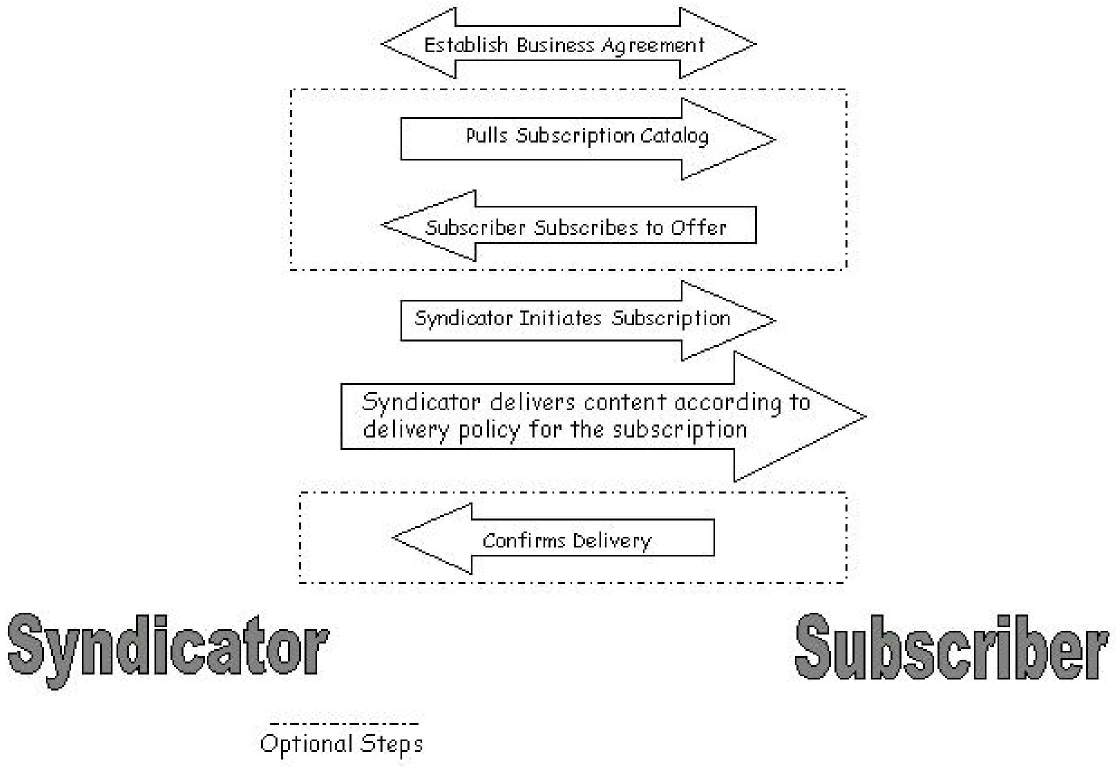
Establish Business Agreement
Pulls Subscription Catalog
Subscriber Subscribes to Offer
Syndicator Initiates Subscription
Syndicator delivers content according to delivery policy for the subscription
Confirms Delivery
Syndicator
Subscriber
Optional Steps

Figure 3.2 Full ICE capabilities

3.3.3 Optional ICE Extensions

The ICE Authoring Group chose to remove a number of capabilities of ICE 2.0 specification to simplify the specification. These capabilities will be detailed in additional specifications that go beyond Full ICE as defined by this document. It will be the responsibility of implementers to take these additional specifications into account when developing syndication solutions.