The Information and Content Exchange (ICE) Protocol

Working Draft Version 2.0

December 1, 2003

This version

http://www.icestandard.org/Spec/SPEC-ICE-2.0.html

Latest version

http://www.icestandard.org/Spec/SPEC-ICE1.1.html

Previous version

http://www.icestandard.org/Spec/SPEC-ICE1.01.html

Latest version eratta

http://www.icestandard.org/Spec/ERATTA-ICE1.1.html

Editors:

Jay Brodsky, Tribune Media Services

Marco Carrer, Oracle Corporation

Dianne Kennedy, IDEAlliance

Daniel Koger, independent consultant

Richard Martin, Active Data Exchange

Laird Popkin, Warner Music Group

Status of this Document

This document is a draft recommended specification. It represents a significant step towards a stable specification suitable for widespread dissemination and implementation. It has NIETHER been reviewed NOR approved by the ICE-AG or the Board of Directors of IDEAlliance. The ICE Authoring Group intends to submit this update of the ICE specification to the W3C to replace the original ICE Note specifying version 1.0.

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.

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

Chapter 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 syndication of content. RSS was designed to be simple to use and inexpensive to implement. RSS has proven quite useful for the syndication of free content, but remains limited in its ability to enforce business rules in the content syndication environment. ICE, on the other hand, was developed by industry content-providers and software vendors to automate the scheduled, reliable, secure redistribution of any 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.

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.

1.3 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.

1.4.4 Use of HTTP:GET as a Transport Mechanism

In addition to the specification of ICE 2.0 with an explicit binding to SOAP, the use of ICE in an asynchronous communications environment dictates that ICE 2.0 also enable the use of `HTTP:GET` as a transport mechanism. 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. The `HTTP:GET` is the simplest form of ICE 2.0 transport and is the only form of ICE transport allowed for Basic ICE conformance.

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 *Extending the ICE Protocol*

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 `<text>` element of `business-term` element or in the `<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 Namespaces

In order to enable interoperability in the Web services environment, ICE 2.0 uses the following namespaces:

- `xmlns:icemes = "http://icestandard.org/ICE/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"`

1.4.9 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 simple datatypes schema is `http://www.icestandard.org/Spec/V20/schema/ice-simpledatatypes.xsd`. This is discussed in more detail in *2.6 ICE Simple Datatypes* and is found in their entirety in Appendix **.

1.4.10 ICE WSDL Scripts

In addition to the schema definitions for ICE 2.0, WSDL scripts are required to 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-full.wsdl`
- `http://www.icestandard.org/Spec/v20/wsdl/ice-syndicator-full.wsdl`

ICE WSDL scripts are found in their entirety in Appendix **.

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. The details of a basic ICE implementation will be discussed in Chapter 4. The details of a full ICE implementation will be discussed in Chapter 5.

- *Chapter 4* describes a basic ICE implementation. This section provides a detailed description of basic protocol operations.
- *Chapter 5* describes a full ICE implementation. This section provides a detailed description of complete protocol operations.
- *Chapter 6* describes how to extend the ICE protocol. This section provides details about how XML Namespaces can be used to extend the ICE protocol.

1.6 Conventions

This document contains a number of constructs that are identified and numbered within a chapter. These structures include:

- 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.

Chapter 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 `<get-package` request to a Syndicator, the Subscriber is the Requester. When a Syndicator pushes a `<package` request to a Subscriber, the Syndicator becomes the Requester and waits for a `<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 HTTP:GET

In Basic ICE all messages are initiated with an `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 1.0 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.

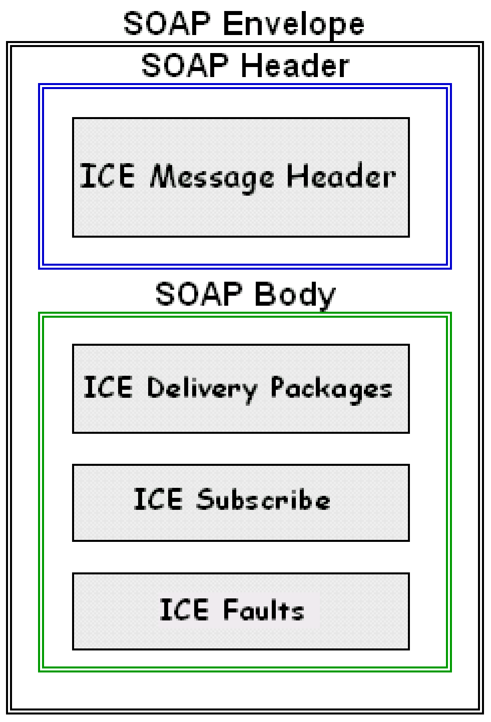


Figure 2.1. ICE carried by SOAP

For either Full ICE or Basic ICE, content is encoded in an ICE/SOAP format. This is simply an XML file where SOAP and ICE XML tags are used to wrap the content and ICE faults that may be sent by the Syndicator. An example of an ICE/SOAP message follows.

```
<?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 table below 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 above and use them to track and signal specific states in the syndication relationship. The table below describes the attributes that contain the identifiers described in the table above.

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 found at `http://www.icestandard.org/ICE/2002/simpledatatypes.xsd`. 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 http://www.icestandard.org/ICE/Spec/V20/schema/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 PING and OK diagnostic messages. See Figure 2.2.

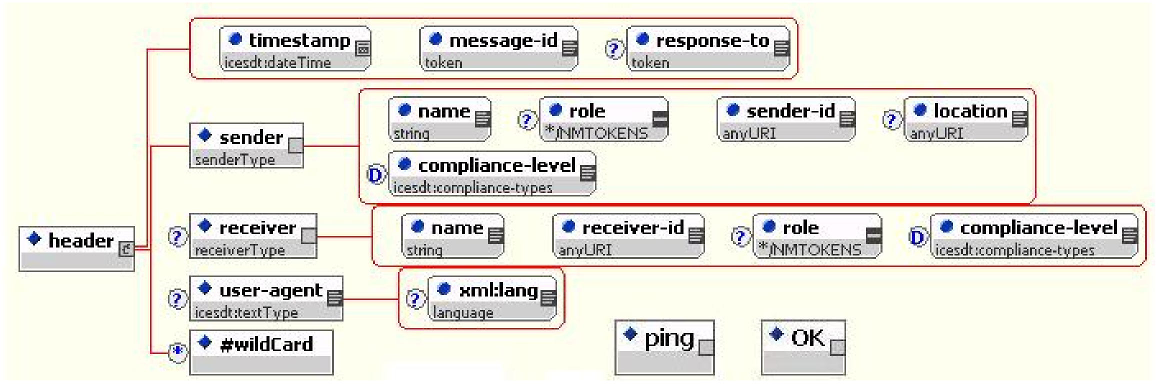


Figure 2.2. ICE message XSD

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 http://www.icestandard.org/ICE/Spec/V20/schema/ice-delivery.xsd as 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.

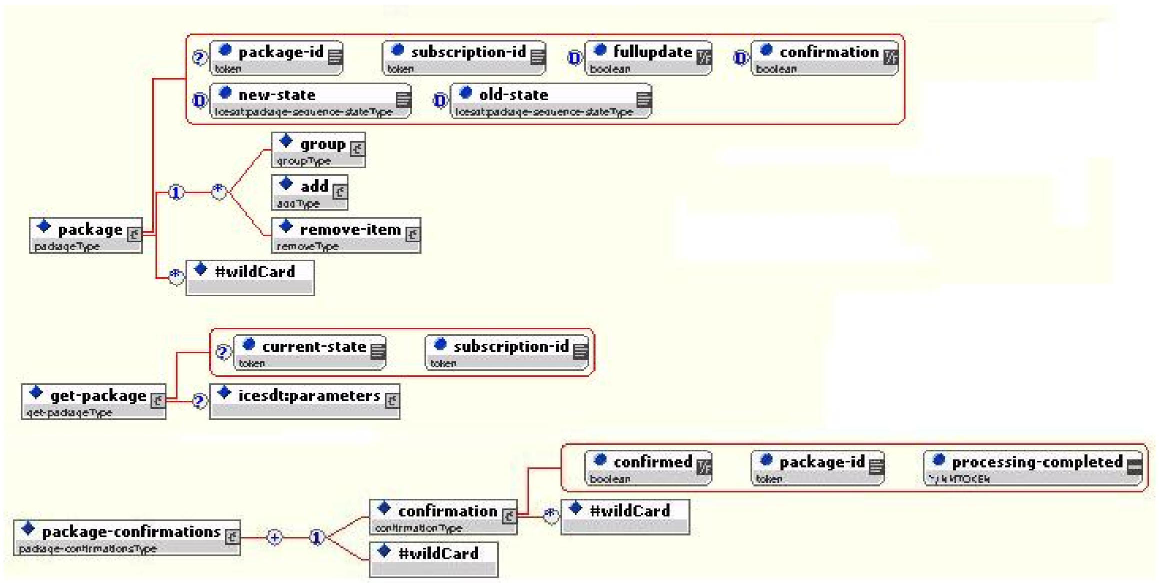


Figure 2.3. ICE delivery elements

ICE delivery is most often made up of packages. Packages may directly contain content from another XML namespace, indicated by `#wildcard` or `<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 `<get-packages` request and a `<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 http://www.icestandard.org/ICE//Spec/V20/schema/ice-subscribe.xsd as the “`icesub`” namespace. See Figure 2.4.

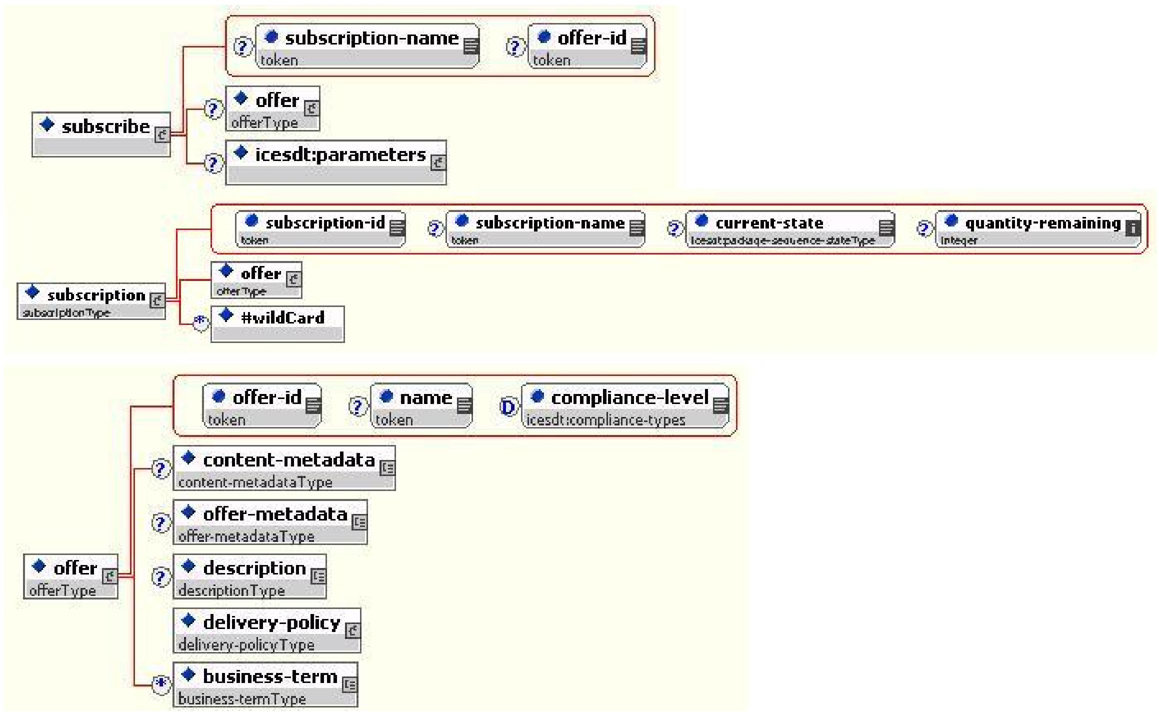


Figure 2.4. ICE subscription elements

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 policy rule defines how and when content will be delivered. See Figure 2.5.

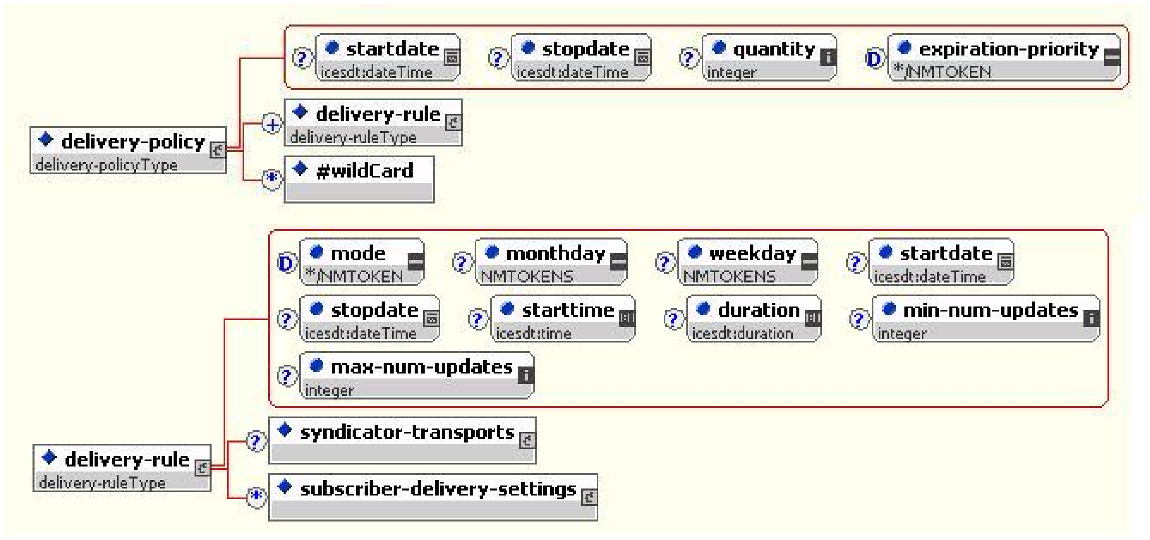


Figure 2.5. ICE Delivery Rule Structure

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.

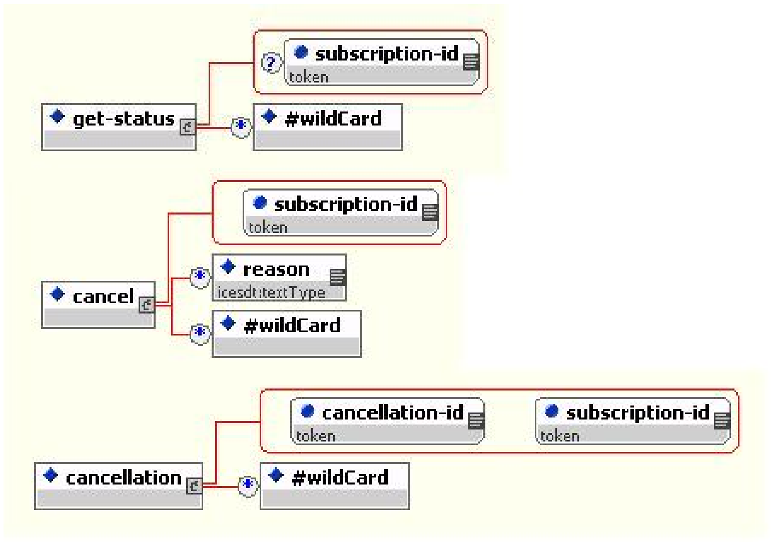


Figure 2.6. ICE subscription management elements

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.

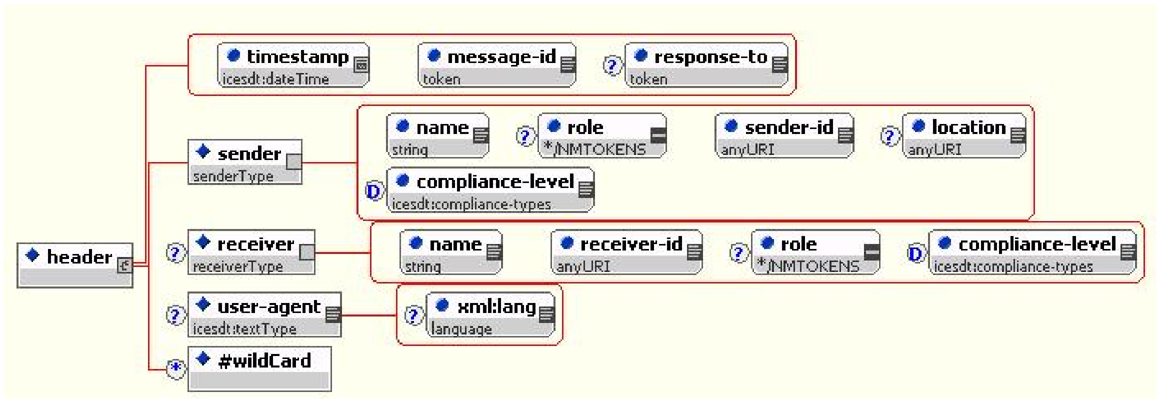


Figure 2.7 ICE Message Header Structure

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 ICE status codes is described by the following schema fragment in ICE 2.0:

```
<element name = "status-code">
  <complexType>
   <attribute name = "code" use = "required" type =
"positiveInteger"/>
   <attribute name = "message-id" use = "required" type =
"token"/>
   <attribute name = "subscription-id" use = "required"
        type = "token"/>
   <attribute name = "location" type = "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, these must 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.

Chapter 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.

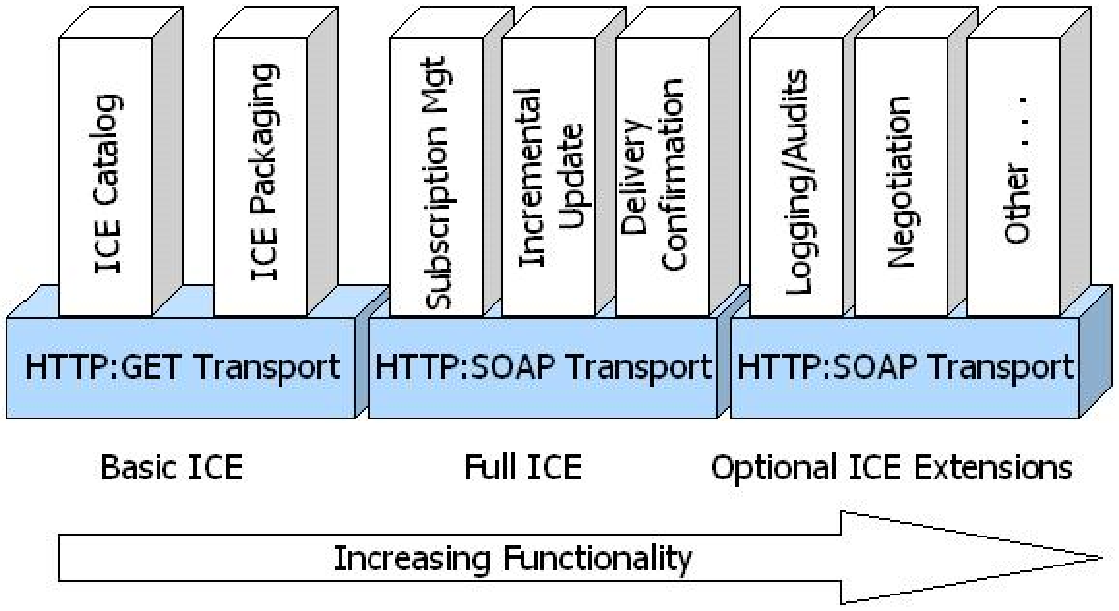


Figure 3.1. ICE modules by increasing functionality

3.1.1 HTTP:GET (REST) Transport

The first module of ICE is the simple `HTTP:GET` transport. The binding for this transport does not need to be defined by a WSDL script. The `HTTP:GET` mechanism enables a very simple form of ICE syndication this is known as *Basic ICE*. 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. 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 *Chapter 5 Full ICE*.

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 **.

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 Chapter 6.

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 Chapter 6.

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:

- `<package` (this is limited to a single package)
- `<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 Chapter 4 for Basic ICE features/modules.

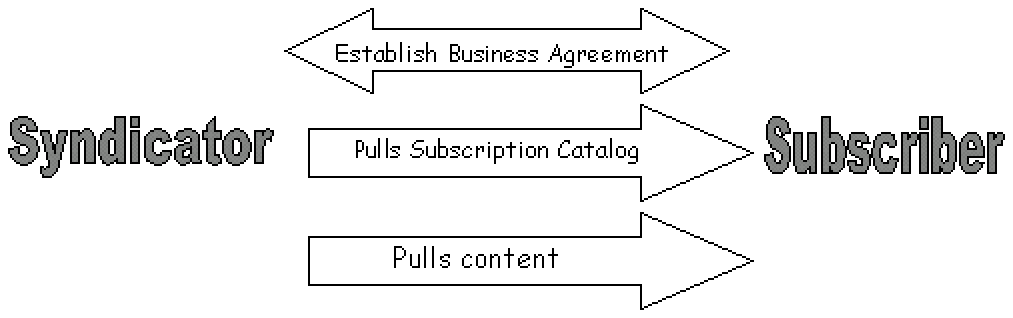


Figure 3.1 Basic ICE capabilities

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:

- `<subscribe`
- `<subscription`
- `<cancel`
- `<cancellation`
- `<get-status`
- `<status`
- `<package-confirmations`

Refer to Chapter 5 for Full ICE features/modules.

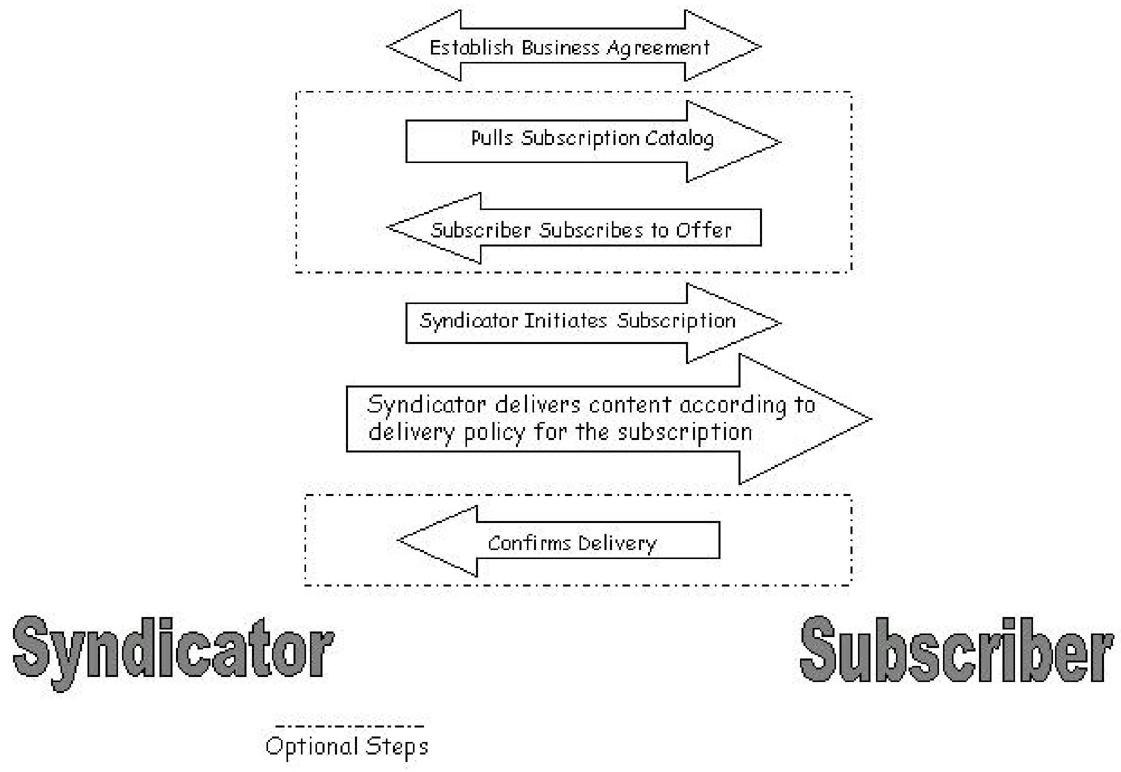

Establish Business Agreement
Pulls Subscription Catalog
Subscriber Subscribes to Offer
Syndicator Initiates Subscription
Syndicator delivers content according to delivery policy for the subscription
Confirms Delivery
Syndicator
Subscriber
Optional Steps

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.

Chapter 4. Basic ICE

4.1 Overview

Due to the nature of the content syndication business, it is important for ICE to support Subscriber implementations of varying levels of sophistication. In the most general case, a Subscriber is a sophisticated server implementation capable of not only sending ICE requests, but also receiving communications initiated by the Syndicator at any time, such as the "push" of new content. A Full ICE Subscriber has an ICE server running at all times. ICE also supports the concept of a *Basic ICE* implementation. This is an implementation where the Subscriber can initiate communicates (e.g. polling for updates) but does not have a persistent server available to receive messages. It is expected that a Basic ICE Subscriber is run on demand, either by a user or by an automated script. Thus, in a Basic ICE implementation, communication is out-of-band.

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. 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 the Subscriber to establish a “listener” for push messages.

4.2 A Basic ICE Scenario

Let’s look at a step-by-step example of a simple transaction between a Syndicator and a Subscriber. See Figure 4.1.

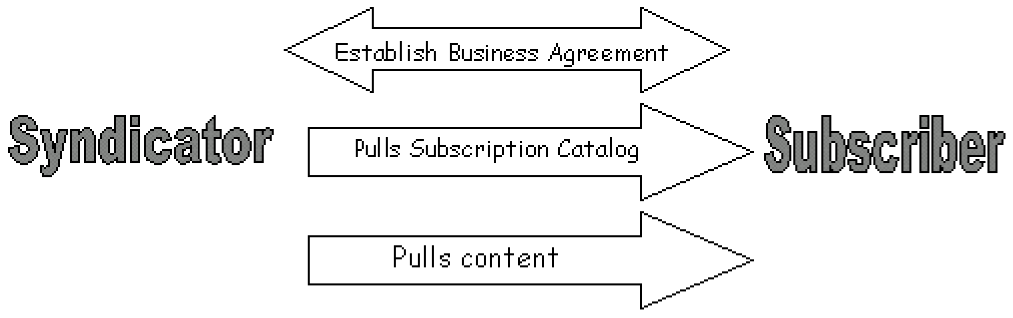


Figure 4.1 A Basic ICE Scenario

4.2.1 Syndicator and Subscriber Set up a Business Agreement

Syndication relationships begin with a business agreement. The business agreement negotiation happens *outside* ICE and can involve person-to-person discussion, legal review, and contracts. Because Basic ICE has no subscription management features, these parameters may be discussed as part of the business agreement. We believe Basic ICE will most often be used in the scenario where the Syndicator is making content freely available to anyone. Hence subscription management features are not required.

4.2.2 Syndicator Makes "Catalog" Available

In ICE Version 1.0, the ICE protocol defined a set of messages to deliver a catalog of offers from the Syndicator to the Subscriber. In ICE 2.0, these messages were replaced by using the Version 2.0 content-syndication mechanism directly to deliver a package of offers where the `subscription-id="1"`. This ICE package should be placed at `http://<server-url>/get-package/1`.

The following example shows a package with a Basic ICE offer provided by a Syndicator.

```
<icedel:package
xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
                fullupdate="true"
                package-id="1"
                subscription-id="1">
  <icedel:add>
    <icedel:metadata
 item-type="http://icestandard.org/ICE/V20/item-type/offer"
             content-type="text/xml"/>
    <icedel:item>
      <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
       <icesub:delivery-policy>
          <icesub:delivery-rule/>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

Note: In this offer everything is left to default including the mode on the delivery rule. ICE 2.0 was designed so that all defaults support Basic ICE. The transfer protocol will be "`http:get`" and the packaging will be "`ice`". The pull will be made from the same location from which the catalog was pulled.

4.2.3 Subscriber "Gets" ICE Catalog

In ICE 1.0, messages were provided for retrieving the ICE catalog. In Basic ICE 2.0, the catalog is retrieved using the same mechanism as is used to retrieve other content. By convention, if the `subscription-id="1"` the package is made up of ICE offers.

4.2.4 Subscriber "Gets" Content

To retrieve content in Basic ICE, the Subscriber does an `HTTP:GET` on the URL specified in the offer that is selected by the Subscriber.

Basic ICE does not enforce complex delivery rules. All content is *pulled from* the Syndicator in Basic ICE. The Subscriber pulls ICE content encoded in an ICE/SOAP format. This is simply an XML file where SOAP and ICE are used to wrap the content and ICE status codes that may be sent by the Syndicator. The Subscriber can treat this package as it would treat any XML-encoded file. It is not necessary that the Subscriber have SOAP capabilities to receive content or ICE status codes.

An example of what the Subscriber gets from the Syndicator's URL follows.

```
<?xml version="1.0" ?>
<env:Envelope
xmlns:env='http://www.w3.org/2002/12/soapenvelope'>
  <env:Header>
    <icemes:header
  xmlns:icemes="http://icestandard.org/ICE/2002/message"
        timestamp="2003-03-03T00:00:00Z"
         message-id="m0056">
      <icemes:sender name="mycompany"
   role="http://icestandard.org/ICE/2002/role/syndicator"
         sender-id="http://www.xxyz.org"/>
    </icemes:header>
  </env:Header>
  <env:Body>
    <icedel:package
  xmlns:icedel="http://icestandard.org/ICE/2002/delivery"
       fullupdate="true"
       package-id="12"
       subscription-id="3">
      <icedel:add>
        <icedel:item-ref>
          <icedel:reference
             url="http://mysite.com/text.htm"/>
        </icedel:item-ref>
      </icedel:add>
    </icedel:package>
  </env:Body>
</env:Envelope>
```

4.3 Transport and Messaging

Two entities are involved in ICE transport and messaging. The Syndicator produces content that is pulled by Subscribers. The philosophy behind Basic ICE is to enable a very simple form of syndication that does not require sophisticated processing by either the Syndicator or the Subscriber. In Basic ICE, all messages/packages from the Syndicator are accessible from a URL and the Subscriber uses an `HTTP:GET` to retrieve messages. This implies that the Subscriber does not have to have a SOAP-enabled server to receive "push" content from the Syndicator. Rather the Subscriber always pulls from the Syndicator. Because there are no end points for either the Syndicator or the Subscriber, a WSDL script for the Basic ICE does not exist.

4.4 Catalogs and Subscription Management.

Basic ICE was designed for publishing information that is available to any interested party. Basic ICE does not support subscription management. So, with Basic ICE, there are no `<icesub:subscribe` or `<icesub:subscription` elements. The Subscriber cannot use `<cancel`. Catalogs of syndication offers, that were uniquely identified by an element type in ICE 1.0, are simply a special type of `<icedel:package` that contains offers (the ICE catalog) and by convention is identified by the `subscription-id="1"`.

4.4.1 Offers

The structure of an offer for Basic ICE is shown in Figure 4.2. Most of the attributes on offer and most of the elements that make up offer are only used in Full ICE and are not shown here. This Chapter will only document those that apply to Basic ICE.

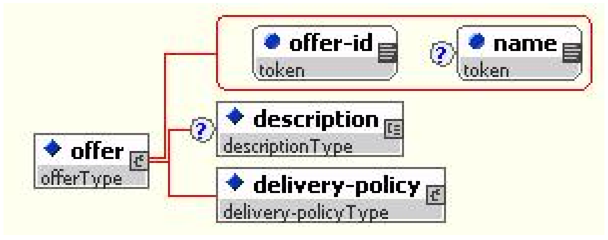


Figure 4.2 Basic ICE Offer Structure

ICE was written so that all defaults are set to support basic ICE. Hence offers in Basic ICE are themselves very basic.

An `<icesub:offer` has the following attributes that will be used for Basic ICE:

- **offer-id**
  Required. This is a string. It is an identifier that MUST be unique across all catalog offers from a Syndicator. Its function is to clearly identify this offer from all other catalog offers made by a Syndicator.
- **name**
  Optional. This is a string. It is a name that may be used to distinguish subscriptions and offers from other subscriptions or offers. Its intended use is to provide a readable short description of the offer such as, "Julia Child's Contemporary French Cooking Column".

An offer is made up of a several elements that will be used for Basic ICE. These include `<icesub:description` and `<icesub:delivery-policy.`

4.4.1.1 Description

This element is a text field and facilitates the entry of a description of the offer. This simple element is shown in Figure 4.3. Note that the `xml:lang=` attribute on `<icesub:text` enables the specification of language for the text field within `<icesub:description`.

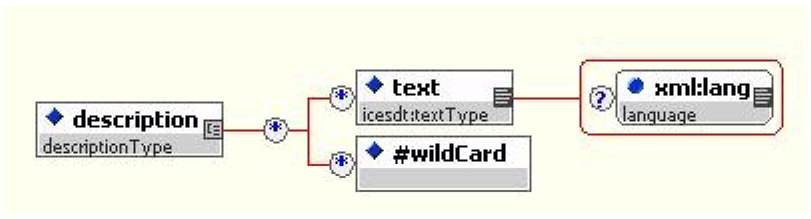


Figure 4.3 Description element structure

The description is useful in Basic ICE because it can help Subscribers understand whether they want to pull the content of the subscription.

4.4.1.2 Delivery Policy

Each subscription offer has one delivery-policy. The delivery policy can determine, for example, the times and dates during which packages can be pulled for a given subscription. Each delivery policy has one or more delivery rules. See Figure 4.4.

The Subscriber must accept the delivery policy within an offer and all required delivery rules within a delivery policy. They can select among optional delivery rules, however.

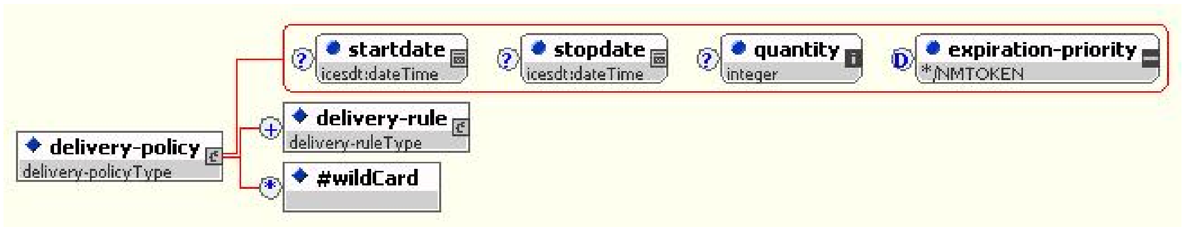


Figure 4.4 Delivery-policy Structure

The `<icesub:delivery-policy` element is defined as the type *delivery-policyType*. This is defined by the following XML schema fragment:

```
<xs:complexType name = "delivery-policyType">
  <xs:sequence>
    <xs:element name = "delivery-rule"
      type = "delivery-ruleType" maxOccurs = "unbounded"/>
    <xs:any namespace = "##other"
      processContents = "lax" minOccurs = "0"
      maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:attribute name = "startdate" type =
"icesdt:dateTime"/>
  <xs:attribute name = "stopdate" type =
"icesdt:dateTime"/>
  <xs:attribute name = "quantity" type = "xs:integer"/>
  <xs:attribute name = "expiration-priority"
     default = "first">
   <xs:simpleType>
    <xs:restriction base = "xs:NMTOKEN">
      <xs:enumeration value = "first"/>
      <xs:enumeration value = "time"/>
      <xs:enumeration value = "quantity"/>
      <xs:enumeration value = "last"/>
      </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:anyAttribute namespace = "##other"
      processContents = "lax"/>
</xs:complexType>
```

The attributes for `<icesub:delivery-policy` that can be used for Basic ICE are:

- **startdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the date and time on which the schedule will start to apply. See the discussion under *ICE Datatypes* for details of what this means. If this attribute is omitted, the schedule will start immediately.
- **stopdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the Date and Time on which the schedule expires. See the

discussion under *ICE Datatypes* for details of what this means. If this attribute is omitted, the schedule never expires (unless superseded in the future).

- **quantity**
  Optional. Datatype is an integer. This attribute specifies the quantity of updates in the subscription
- **expiration-priority**
  Default. This attribute specifies the expiration priority. Values are `first`, `last`, `time`, and `quantity`. The default is `first`.

NOTE: The multiple delivery-rules in a delivery-policy are conceptually joined with "OR" (not "AND"). In other words, the valid delivery times are the union of all the times defined by each rule in the delivery policy.

4.4.1.2.1 Delivery Rule

Each `<icesub:delivery-policy` is made up of one or more delivery rules. Only certain elements and attributes of a delivery rule makes sense for Basic ICE. These structures are shown in the reduced diagram of `<icesub:delivery-rule` in the Figure 4.5. Each delivery rule in Basic ICE is made up of one or more `<icesub:transport` elements.

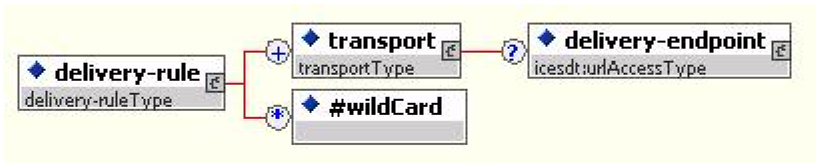


Figure 4.5 Basic ICE Delivery-rule Structure

4.4.1.2.2 Transport

The `<icesub:delivery-rule` is made up of one or more `<icesub:transport.` Transports are specified when the Syndicator makes an offer. This element provides a mechanism for the Syndicator to indicate the delivery transports for the `<icesub:offer`. You can see the makeup of a Basic ICE `<icesub:transport` in the Figure 4.6:

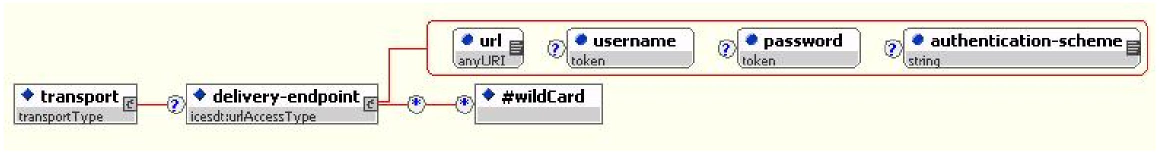


Figure 4.6 Basic ICE Transport Structure

The <icesub:transport is made up of an optional delivery endpoint. See Figure 4.6. In Basic ICE, when the Syndicator is offering content in “`pull`” mode, a delivery endpoint for that pull can be specified by using `<icesub:delivery-endpoint`.

The `<icesub:delivery-endpoint` has 4 attributes. These include:

- **url**
  Required. This attribute specifies the URL for push delivery. The datatype is `anyURI.`
- **username**
  Optional. This attribute specifies an optional username that may be required to access URL for push delivery.
- **password**
  Optional. This attribute specifies an optional password that may be required to access URL for push delivery.
- **user-authentication**
  Optional. This attribute specifies the optional user authentication scheme. It is a string with enumerated values of "`basic`" and "`digest`". There is no default

4.4.1.3 Example Basic ICE Offer

Basic ICE is a simple "`pull`" of content by the Subscriber. The following example shows a Basic ICE offer, using the built-in defaults for Basic ICE.

Syndicator transports were left to default. Delivery settings are not provided because the assumption is that the protocol will be "`http:get`" and the packaging will be "`ice`" and The pull will be made from the URL specified by the endpoint.

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
   <icedel:item>
      <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    name="offName2"
    offered=”off2”>
      <icesub:description>
           new stories headlines and abstracts
      </icesub:description>
       <icesub:delivery-policy>
          <icesub:delivery-rule>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

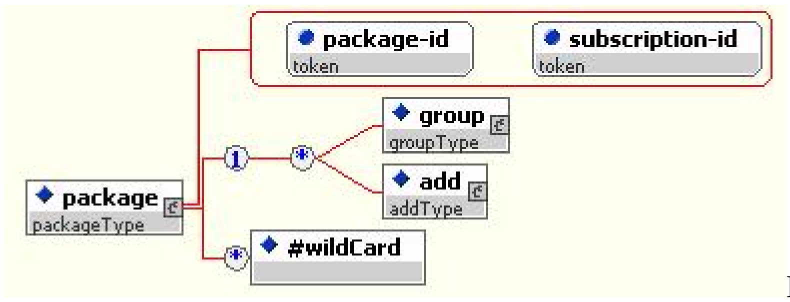
NOTE: You can recognize that this package contains a subscription offer because the subscription-id is set to "1".

4.5 Packages and Delivery

Basic ICE, like Full ICE, supports the delivery of packages. In Basic ICE, the delivery is simply the act of the Syndicator placing the content in a SOAP/ICE XML document at the URL specified in the offer. The XML definition for packages is found at http://www.icestandard.org/Spec/V20/schema/ice-delivery.xsd.

The ICE package is made up of three elements. See Figure 4.2.NOTE: Even the though the

ICE package is made up of three elements, only two of these make sense for Basic ICE. In Basic ICE it is reasonable that the Syndicator delivers a group of items or adds a single item to the subscription content. Since each package contains a full update in Basic ICE, removing items will never be used.



Ba

Figure 4.2 Basic ICE Package Structure

4.5.1 Package Attributes

There are several attributes on package that are meaningful for Basic ICE:

- **package-id**
  Required. Identifies the package within the scope of a subscription. The Syndicator assigns the `package-id`.
- **subscription-id**
  Required. In Basic ICE, the subscription-id is the unique id of the content feed and is used by all subscribers. The Syndicator assigns the `subscription-id`.

4.5.2 Package Elements

The ICE package is made up of 3 elements. An ICE `<icedel:package` describes a set of content operations: additions, removals, and a group of additions and/or removals. The remove operation is specified using the `<icedel:remove-item` element is not used in Basic ICE because this functionality facilitates delivery of incremental updates. The content additions contain the content that needs to be added or updated and are specified using the `<icedel:item` and `<icedel:item-ref` elements. The `<icedel:group` element

allows the Syndicator to associate the content specified using the `<icedel:item` elements together. For example, in the syndication of restaurant reviews, each review may consist of different types of content such as an HTML file and two graphic files. These three files could be contained within three `<icedel:item` elements and grouped together in an ICE `<icedel:group` as a single restaurant review. Likewise, unrelated content can be specified in a `<icedel:package` by just using the `<icedel:add` and then `<item` elements without an intervening `<icedel:group`. The `<icedel:item` and `<icedel:item-ref` elements distinguish themselves by the way they contain the content. The `<icedel:item` element is used to contain content directly in the delivered content. The `<icedel:item-ref` element is used to distribute an indirect reference to the actual content. Note that the `#wildcard` allows for insertion of content from any namespace.

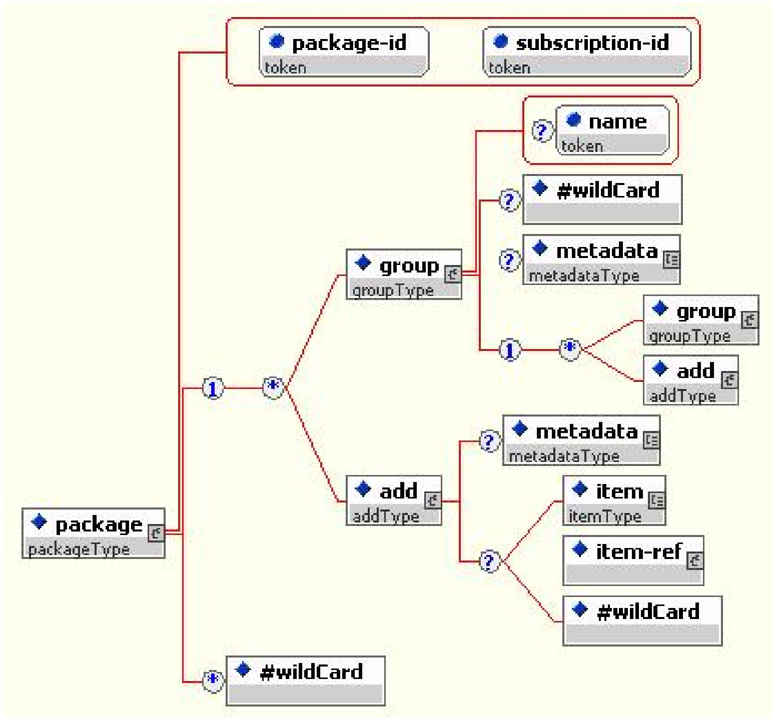


Figure 4.3 Package elements for Basic ICE

4.5.2.1 Group

The `<icedel:group` is a container element that can be used to group content items being added or removed. It also enables the attachment of metadata to a group of content items.

Attributes on `<icedel:group` include:

- **name**
  Optional. This attribute specifies a name for the item group that can be uwsed to identify that group within a package.

4.5.2.2 Metadata

The `<icedel:metadata` element enables the entry of metadata on `<icedel:group` and `<icedel:add` by using its attributes and a description field. See Figure 4.7.

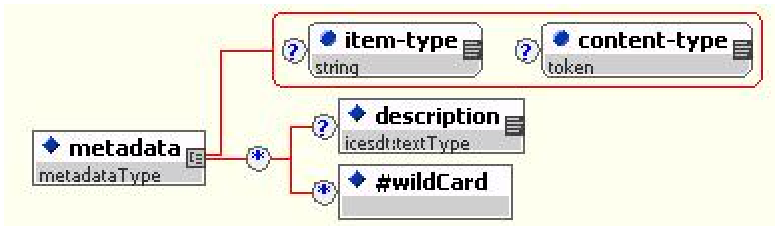


Figure 4.7 Basic ICE Metadata

Because the attributes on the metadata element are set to support Basic ICE, only the content-type and item-type attributes may be used.

- **content-type**
  Optional. This attribute enables the specification of the type of content such as "news."
- **item-type**
  Optional. This attribute specifies a URI that identifies what type of item this is. For example the value of item-type may be `http://icestandard.org/ICE/V20/item-type/offer`.

> Note: For Basic ICE, the requirement is that content will be used in its entirety and that the content may not be edited.

4.5.2.3 Add

The `<icedel:add` element is used to add new content according to the delivery policy of the subscription. It enables the attachment of metadata to the content being added. The `<icedel:add` enables content to be directly included in the message by using the `<icedel:item` element, an indirect reference to content using `<icedel:item-ref` mechanism.

The structure of `<icedel:add` is shown in Figure 4.8.

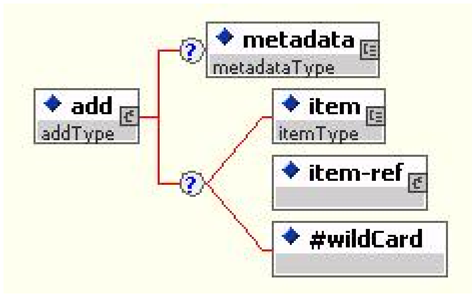


Figure 4.8 Add element structure

4.5.2.4 Item

The `<icedel:item` element directly carries content from the Syndicator to the Subscriber. Each item has a unique item-id. In addition, the `<icedel:item` can carry the ICE element `<icesub:offer`.

4.5.2.5 Item-Ref

The `<icedel:item-ref` element references Syndicator content. The `<icedel:item-ref` structure is shown in Figure 4.9. It is made up of a single `<icedel:reference` element. This means that for each reference, an `<item-ref` element must be used.

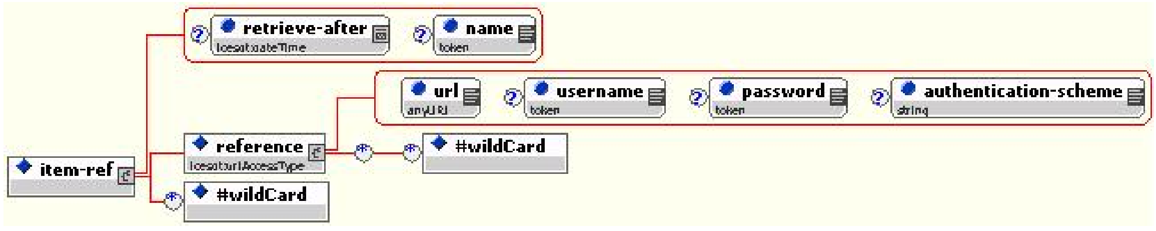


Figure 4.9 Item-ref structure

The `<item-ref` element has two attributes:

retrieve-after
Optional. This attribute specifies a time after which the item can be retrieved. It is specified in the `icesdt:dateTime` format.

name
Optional. This attribute specifies the item name that can be used as a transient identifier within a group or add.

4.5.2.4 Reference

The `<icedel:reference` element is used to reference the content of the `<icedel:item-ref` element. The reference element is empty (with the exception of any wildcard content). The attributes carry the information for this element.

The `<icedel:reference` element has four attributes:

- **url**
  Required. This attribute specifies the URL from which the content can be retrieved.
- **username**
  Optional. This attribute specifies the username for retrieving the content if a login is required.
- **password**
  Optional. This attribute specifies the password for retrieving the content if a login is required.

- **authentificationscheme**
  Optional. This attribute specifies the authentification scheme for retrieving the content if this is required.

Chapter 5. Full ICE

5.1 Overview

Basic ICE provides for a very simple syndication model where the Subscriber does not have an ICE server running constantly and polls for content as required. But, when more robust syndication functionality is required, *Full ICE* is appropriate. Full ICE extends Basic ICE functionality to add subscription management services as well as other advanced capabilities such as “push” delivery.

Full ICE also differs from Basic ICE in that the Subscriber is a sophisticated server implementation capable of not only sending ICE requests, but also receiving communications initiated by the Syndicator, such as the "push" of new content. In a Full ICE implementation both the Syndicator and Subscriber have an ICE server running at all times. Each must support SOAP transport bindings as well as subscription management capabilities.

Because Full ICE is an extension of Basic ICE, a Basic ICE implementation can talk to a Full ICE implementation, but without the advantages of Full ICE.

5.2 A Full ICE Scenario

Let’s look at a step-by-step example of a simple transaction between a Syndicator and a Subscriber in a familiar industry. The Syndicator, the Best Code Company, a software developer, sets up and delivers a subscription to Tech News, a trade journal for the high technology industry. See Figure 5.1.

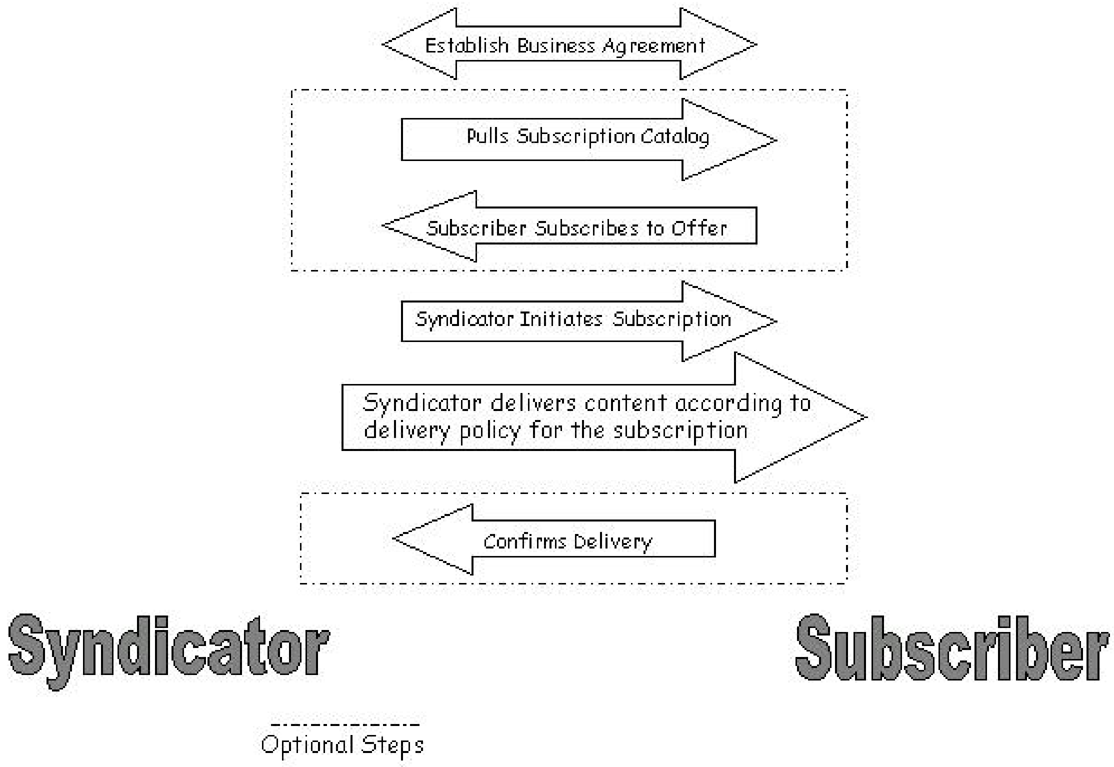

Establish Business Agreement
Pulls Subscription Catalog
Subscriber Subscribes to Offer
Syndicator Initiates Subscription
Syndicator delivers content according to delivery policy for the subscription
Confirms Delivery
Syndicator
Subscriber
Optional Steps

Figure 5.1 Full ICE Scenario

5.2.1 Syndicator and Subscriber Set up a Business Agreement

Syndication relationships begin with a business agreement. Best Code and Tech News agree on such terms as payment issues, usage rights, and subscription lifetime. The business agreement negotiation happens *outside* ICE and can involve person-to-person discussion, legal review, and contracts. Alternatively, a Syndicator could standardize and automate these terms.

5.2.2 Syndicator and Subscriber Set up a Subscription

Once the business agreement is in place, ICE comes into play as Best Code and Tech News start exchanging ICE messages to establish a subscription and begin content delivery.

5.2.2.1 Subscriber Receives Packages of Subscription Offers

In order to view a catalog of subscription offers, Tech News goes to the website of Best Code where the ICE Syndicator’s end point is listed. The Subscriber may also use the discovery mechanism of *Universal Description, Discovery, and Integration (UDDI)* to find the Syndicator’s end point. UDDI represents a set of protocols and a public directory for the registration and lookup of web services specified by UDDI.org. The Subscriber then requests a package of subscription offers using `<icedel:get-packages><icedel:get-package`. By convention, the subscription with the `subscription-id=″1″` returns a Syndicator’s catalog of subscription offers.

5.2.2.2 Subscriber Sends a Request to Subscribe to the Offer

Tech News thinks the press releases are exciting stuff and promptly asks to sign up for the subscription offer. It agrees to pull the content from Best Code’s site and with `maximum-update-interval="P300S"`, which means that the subscriber must check at least every five minutes

5.2.2.3 Syndicator Accepts Request and Responds with Subscription Message

Best Code indicates that it has issued a subscription for Tech News by returning the `<icesub:subscription` message. Best Code gives Tech News a unique subscription-id number and also returns the details of the offer in order to confirm the delivery method.

5.2.3 Subscriber Receives Content

Once the subscription is set up, Tech News is ready to receive content. Tech News starts by asking for new content. Best Code has chosen to take advantage of the Full ICE incremental update capability. This means that the updates contain only changes to the content in the subscription. These changes can add new content and can also include requests to remove outdated content. In this way, Best Code can control the precise content for that subscription on the Tech News site. Together with the actual content, the messages may also specify other subscription parameters such as effective date and expiration date.

5.2.3.1 Subscriber Requests Initial Subscription Content

Tech News uses `<icedel:get-package` to ask for subscription content. The `current-state="ICE-INITIAL"` indicates that this is an initial request for this subscription, which alerts Best Code to download the full content.

5.2.3.2 Syndicator Responds with Full Content of Subscription

Now Best Code delivers the content of its subscription, consisting of an ICE package with a press release and a video file. The press release is part of the package. It is the content of an ICE item element. The video file, however, is not actually in the package. Instead, its location is given in the URL attribute of an ICE `<icedel:item-ref` element. This serves as a pointer to the content and is an alternative to sending the content within the ICE message.

The ICE package element also conveys other information. For example, `editable="true"` gives Tech News permission to edit the content, while `new-state="2"` establishes the state of the subscription. The next time Tech News requests content, it will receive only content added or changed since this delivery, instead of receiving the entire content load all over again.

5.2.3.3 Subscriber Confirms Delivery

If requested by the Syndicator, the Subscriber will return confirmation of content delivery each time content is updated with the `<icedel:package-confirmations` message.

5.2.3.4 Variations on the Full ICE Scenario

Figure 5.1 shows the Full ICE subscription model. Note that an ICE subscription always begins with an out-of-bank business agreement between the Syndicator and the Subscriber. Several steps within the subscription model are optional, depending upon the business agreement. See steps marked with dotted lines in Figure 5.1. For example, the subscription might be initiated without the use of a catalog of offers. In this case the Syndicator can simply issues an `<icesub:subscription` message, provide a unique subscription identifier and begin delivering content.

5.3 Transport and Messaging

Two entities are involved in ICE transport and messaging. The *Syndicator* produces content that is delivered to *Subscribers.* The philosophy behind Full ICE is to enable syndication as a Web service. A Full ICE implementation implements all the features of the ICE 2.0 specification and supports SOAP transport bindings. Full ICE requiresthat the Syndicator and the Subscriber will establish a "listener" to receive messages and that either side will be able to request/respond and send/receive.

5.3.1 SOAP Binding

Because Full ICE supports SOAP transport bindings, two WSDL scripts are included in the ICE 2.0 Specification. These scripts define the transport for the Full ICE Syndicator and for the Full ICE Subscriber. The WSDL scripts can be found, in their entirety in Appendix **.

```
<!--      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"/>
    <operation>
       OPERATIONS GO HERE
    </operation>    . . .
  </binding>
```

5.3.2 Integrated ICE/SOAP Message

As was discussed in Chapter 2, 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 within the SOAP header and the ICE delivery and subscription mechanisms were designed to be enclosed in the SOAP body. See Figure 2.1.

5.4 Subscription Management

Basic ICE does not support subscription management. Only with Full ICE conformance may a Syndicator manage the subscriptions and data feeds to individual subscribers. Of course, Full ICE Syndicators may also provide public syndication feeds that are freely available to all Subscribers. But Full ICE was designed to support the business management of content syndication.

5.4.1 Subscription Establishment Overview

Subscription relationships in ICE usually begin with a request by the Subscriber to obtain a *catalog* of *subscription offers* from the Syndicator. As already described, prior to the Subscriber making this request, the Subscriber and the Syndicator have already engaged in discussions regarding licensing terms, payment options, and other business considerations. This happens outside of the ICE protocol. Once the parties agree that they wish to have a content exchange relationship, the ICE process begins.

A typical sequence of events is:

1. A user (technical manager, engineer, etc.) at the Syndicator site creates a new Subscriber account using the ICE software on the Syndicator's system. This operation is not defined by the protocol; it is a property of the tools used by the Syndicator.
2. The Syndicator tells the Subscriber what URL to use for ICE communication. It is likely that this URL will be under access control, and the Syndicator will communicate the necessary authentication data to the Subscriber using an out-of-band mechanism.
3. ICE protocol operations are now ready to begin: the Subscriber will authenticate (if necessary) to the given URL and issue the first ICE request: a `<icedel:get-packages><icedel:get-package subscription-id=”1”>` request for the package containing the catalog of offers.
4. The Syndicator will return a package containing offers.
5. The Subscriber issues a subscription request for an offer using the `<icesub:subscribe` message
6. The Syndicator responds with the `<icesub:subscription` message indicating that the subscription is established and packages can begin to be exchanged.

5.4.2 Get Package of Offers

The first step in the establishment of a subscription is the request from the Subscriber for a package containing a package of offers. This request is initiated by the `<icedel:get-package` message with the `subscription-id=”1”` that is universally known as a package that contains offers available from the Syndicator. See Figure 5.2. Change to get-package as the root!

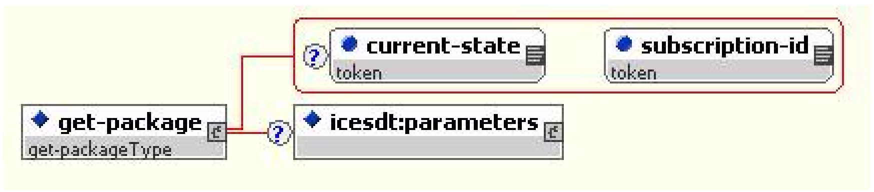


Figure 5.2 Get-package Request Structure

An example of such a request is shown below in its complete ICE/SOAP form:

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env=”http://www.w3.org/2002/12/soap-
envelope”>
  <env:Header>
    <icemes:Header
      xmlns:icedel=http://icestandard.org/ICE/V20/message
      timestamp=”2003-03-03T00:00:00” message-id=”m0056”>
      <icemes:sender name=”mycompany”
        role=”http://icestandard.org//role/subscriber”
        sender-id=”http://www.xxyz.org”/>
    </icemes:Header>
  </env:Header>
  <env:Body>
     <icedel:get-package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
      subscription-id="1"/>
  </env:Body>
<env:Envelope>
```

5.4.3 Offers

The structure of an offer is shown in Figure 5.3. There are numerous attributes on offer. It is made up of optional `<icesub:content-metadata`, `<icesub:offer-metadata`, `<icesub:description`, followed by a required `<icesub:delivery-policy>` that can be followed by an optional `<icesub:business-term, one or more <icesub:required-extension` and other content from the Syndicator’s own schemas (#wildCard).

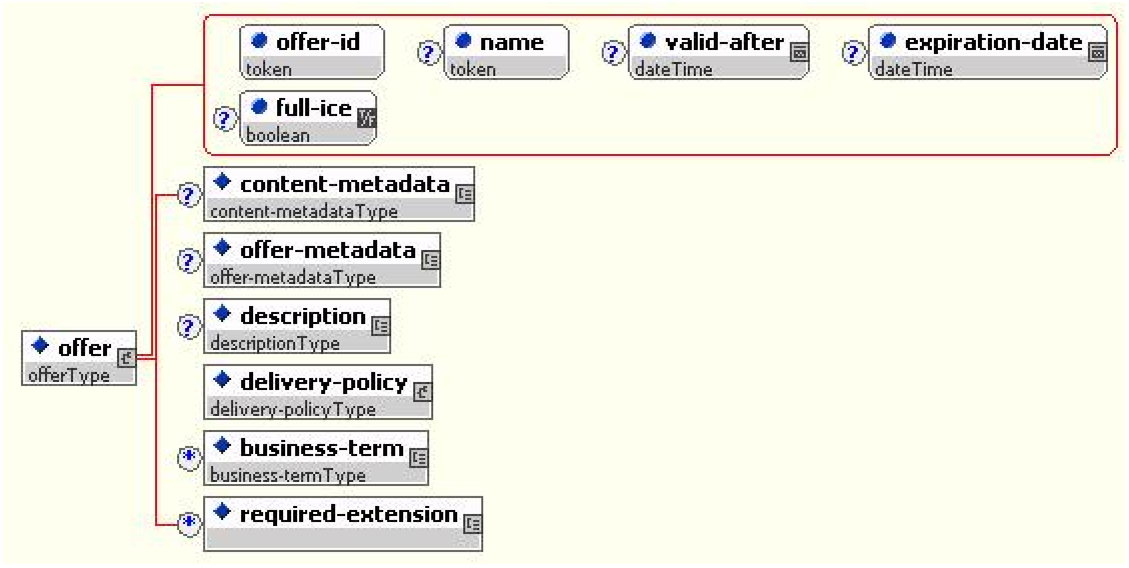


Figure 5.3 ICE Offer structure

The element `<icesub:offer` is defined as the complex type *offerType* as shown in the following fragment of the XML schema:

```
<xs:element name = "offer" type = "offerType"/>
<xs:complexType name = "offerType">
  <xs:sequence>
   <xs:element name = "content-metadata"
      type = "content-metadataType" minOccurs = "0"/>
   <xs:element name = "offer-metadata"
      type = "offer-metadataType" minOccurs = "0"/>
   <xs:element name = "description"
      type = "descriptionType" minOccurs = "0"/>
   <xs:element name = "delivery-policy"
      type = "delivery-policyType"/>
   <xs:element name = "business-term"
      type = "business-termType" minOccurs = "0"
      maxOccurs = "unbounded"/>
   <xs:element name = "required-extension"
      minOccurs = "0" maxOccurs = "unbounded">
    <xs:complexType>
     <xs:complexContent>
      <xs:extension base = "required-extensionType">
       <xs:attribute name = "extension-type"
          use = "required" type = "xs:anyURI"/>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name = "offer-id" use = "required"
     type = "xs:token"/>
  <xs:attribute name = "name" type = "xs:token"/>
  <xs:attribute name = "valid-after" type = "xs:dateTime"/>
  <xs:attribute name = "expiration-date"
     type = "xs:dateTime"/>
  <xs:attribute name = "full-ice" default = "false"
     type = "xs:boolean"/>
  <xs:anyAttribute namespace = "##other"
     processContents = "lax"/>
</xs:complexType>
```

5.4.4 Offer Attributes

An `<icesub:offer` has the following attributes:

- **offer-id**
  Required. This is a string. It is an identifier thatMUST be unique across all catalog offers between a Syndicator and Subscriber. Its function is to clearly identify this offer from all other catalog offers made by a Syndicator to a Subscriber.
- **name**
  Optional. This is a string. It is a name that may be used to distinguish subscriptions and offers from other subscriptions or offers. This is provided for use

by the syndicator or subscriber and has no defined ICE semantics. Its intended use is to provide a readable short description of the offer such as, "Julia Child's Contemporary French Cooking Column".

- **full-ice**
  Default. This is Boolean. The default is set to "false" for Basic ICE.
- **valid-after**
  Optional. This attribute has a dateType datatype. It is used to specify a date when the offer becomes valid and may be accepted.
- **expiration-date**
  Optional. This attribute has a dateType datatype. It is used to specify a date when the offer expires and is no longer valid.

5.4.5 Offer Elements

An offer is made up of a optional elements `<icesub:content-metadata`, `<icesub:offer-metadata`, `<icesub:description`, `<icesub:delivery-policy` `<icesub:business-term` , `<icesub:required-extensions` and allows for the inclusion of content from the Syndicator's own schema (`#wildCard`).

5.4.5.1 Content Metadata

Content-metadata is an element that provides the means for additional metadata that applies to all of the content being offered. The structure of this element is shown in Figure 5.4

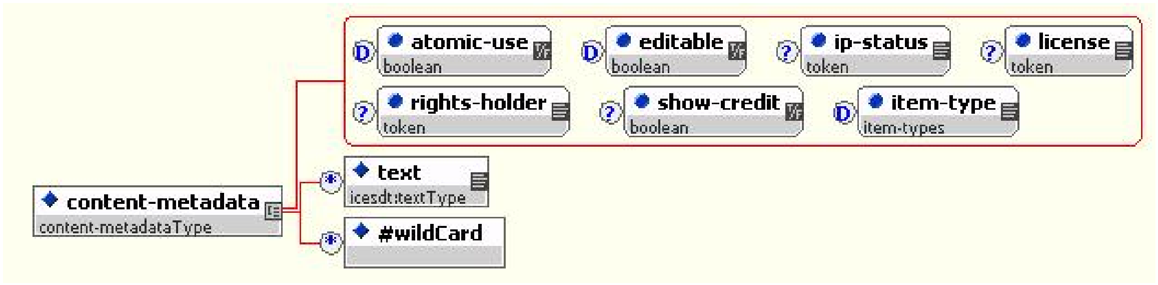


Figure 5.4 Content-metadata Structure

The `<icesub:content-metadata` element provides a mechanism to include additional metadata about the content. The metadata can be entered in the text field or other content metadata from the Syndicator's own schemas (`#wildCard`) can be included. Note that the `xml:lang=` attribute enables the specification of language used in the `<icesub:text` element.In addition, a number of optional metadata fields are provided as content-metadata attributes by the ICE 2.0 specification.

The attributes of `<icesub:content-metadata` include:

- **atomic-use**
  Optional. This is a Boolean. If "`true`", indicates that all information in the subscription must be used together, or not used at all. If "`false`", or unspecified, then the Subscriber is permitted to use subsets of the data in any way they want

(and as permitted by the licensing terms, of course). This flag is meant to be useful as a hint/reminder displayed in a Subscribers ICE tool; ICE cannot enforce it (and, the use of lower case "must" in the above description is intentional; there is no protocol requirement here).

- **ip-status**
  Optional. This is a string describing the intellectual property-rights status of the content. ICE cannot enforce any of these semantics; rather, the intent is that this attribute allows the Syndicator to communicate useful information to the Subscriber ICE tool, which will ideally display this information in some useful presentation form. This attribute **MAY** contain any arbitrary string determined by the Syndicator. ICE defines the following specific string values, and Syndicators **SHOULD** use them as appropriate:

  - `PUBLIC-DOMAIN`
    The content has no licensing restrictions, whatsoever.
  - `FREE-WITH-ACK`
    The content has no licensing restrictions beyond a requirement to display an acknowledgement of the content source.
  - `SEE-LICENSE`
    The content has licensing restrictions as already agreed to in an existing licensing agreement. This is meant to convey the default case.
  - `SEVERE-RESTRICTIONS`
    The content has licensing restrictions that are worthy of special attention. NOTE: it is the intent that this flag would not be used routinely by Syndicators. The intent is that an ICE tool might "red flag" content marked with this attribute and bring it specially to the attention of an administrator on the Subscriber site (this makes more sense when this attribute is attached to package items).
  - `CONFIDENTIAL`
    The content is confidential and must be protected specially.

- **license**
  Optional. Token indicating the license for the content.
- **rights-holder**
  Optional. String describing the original source of the syndication rights.
- **show-credit**
  Optional. This is a Boolean. If `true`, indicates that the Subscriber is explicitly expected to acknowledge the source of the data.
- **editable**
  Optional. This is a Boolean. If `true`, indicates that the Subscriber may edit/alter the content before using it. If `false`, or unspecified, the Subscriber is expected to use the content without any alteration. It has the same "hint" semantics as *atomic-use*.
- **item-type**
  Optional. This attribute is used to specify the type of content item that is being offered. The datatype is a URI that specifies the content type. This attribute was

designed to indicate the datatype of the content of the subscription so that the subscriber will know whether they can process the content of the subscription being offered.

This is an example of `<icesub:content-metadata`:

```
<icesub:content-metadata
      atomic-use="true"
      editable="false"
      ip-status="Free With Acknoledgement"
      rights-holder="Oracle Corporation, 2003"
      show-credit="true"
      item-type="http://icestandard.org/ICE/V20/item-
type/rss2.0"/>
```

NOTE: The item-type attribute in this example is used to specify the “flavor” of RSS being used in the content.

5.4.5.2 Offer Metadata

Offer-metadata is an element that provides the means for additional metadata to be communicated between the parties specific to an offer. The structure of this element is shown in Figure 5.5:

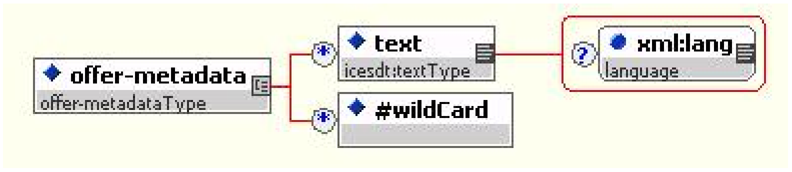


Figure 5.5 Offer metadata structure

The `<icesub:offer-metadata` element provides a mechanism to include additional metadata about the offer. The metadata can be entered in the text field or other content metadata from the Syndicator’s own schemas (#wildCard) can be included.

5.4.5.3 Description

This element is a text field and facilitates the entry of a description of the offer. This simple element is shown in Figure 5.6. Note that the `xml:lang=` attribute on `<icesub:text` enables the specification of language for the text field within `<icesub:description`.

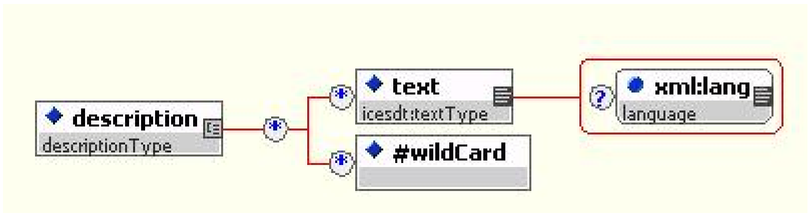


Figure 5.6 Description element structure

5.4.5.4 Delivery Policy

Each subscription offer has one delivery-policy. The delivery policy can determine, for example, the times and dates during which packages can be delivered (push) or requested (pull) for a given subscription. Each delivery policy has one or more delivery rules.

The subscriber must accept the delivery policy within an offer and all required delivery rules within a delivery policy. They can select among optional delivery rules, however.

A delivery-policy has a start date and a stop date attributes, and contains one or more delivery rules. See Figure 5.7 for the delivery-policy structure.

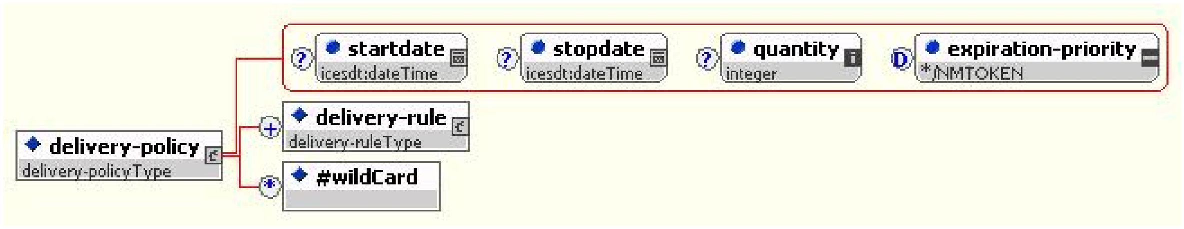


Figure 5.7 Delivery-policy Structure

The `<icesub:delivery-policy` element is defined as the type *delivery-policyType*. This is defined by the following XML schema fragment:

```
<xs:complexType name = "delivery-policyType">
  <xs:sequence>
    <xs:element name = "delivery-rule"
      type = "delivery-ruleType" maxOccurs = "unbounded"/>
    <xs:any namespace = "##other"
      processContents = "lax" minOccurs = "0"
      maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:attribute name = "startdate" type = "icesdt:dateTime"/>
  <xs:attribute name = "stopdate" type = "icesdt:dateTime"/>
  <xs:attribute name = "quantity" type = "xs:integer"/>
  <xs:attribute name = "expiration-priority"
     default = "first">
   <xs:simpleType>
    <xs:restriction base = "xs:NMTOKEN">
      <xs:enumeration value = "first"/>
      <xs:enumeration value = "time"/>
```

```
       <xs:enumeration value = "quantity"/>
       <xs:enumeration value = "last"/>
       </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:anyAttribute namespace = "##other"
       processContents = "lax"/>
</xs:complexType>
```

The attributes for `<icesub:delivery-policy` are:

- **startdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the date and time on which the schedule will start to apply. See the discussion under *ICE Datatypes* for details of what this means. If this attribute is omitted, the schedule will start immediately.
- **stopdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the Date and Time on which the schedule expires. See the discussion under *ICE Datatypes* for details of what this means. If this attribute is omitted, the schedule never expires (unless superseded in the future).
- **quantity**
  Optional. Datatype is an integer. This attribute specifies the quantity of updates in the subscription
- **expiration-priority**
  Default. This attribute specifies the expiration priority. Values are `first, last, time`, and `quantity`. The default is `first`.

NOTE: The multiple delivery-rules in a delivery-policy are conceptually joined with "OR" (not "AND"). In other words, the valid delivery times are the union of all the times defined by each rule in the delivery policy.

5.4.5.4.1 Delivery Rule

Each `<icesub:delivery-policy` is made up of one or more delivery rules. The `<icesub:delivery-rule` can define a window of time during which deliveries can be performed along with other delivery options. Each delivery-rule has a mode of either a push or pull, can define when deliveries can be performed, a start and ending time for the update window, the frequency with which updates can be performed, the count of the number of updates that can be performed and the transport and packaging. In addition, attributes on the delivery rule specify whether updates will be full or incremental and whether this delivery rule is required. You can see the makeup of a `<icesub:delivery-rule` in the Figure 5.8:

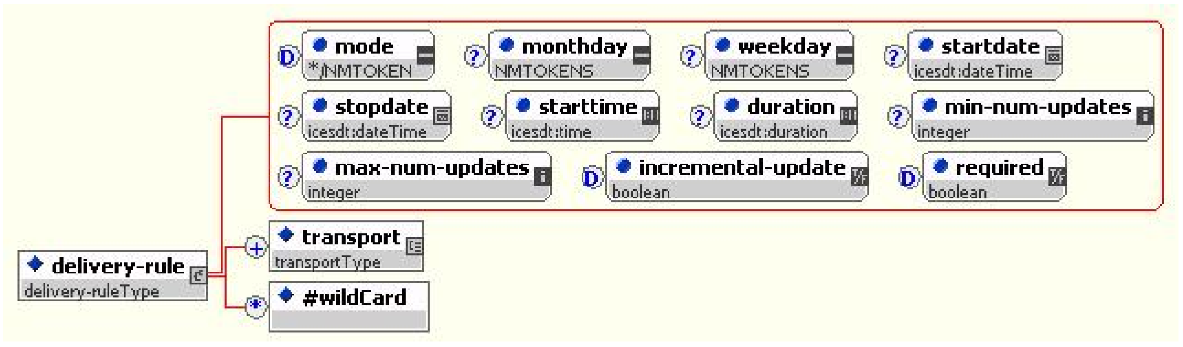


Figure 5.8 ICE delivery-rule structure

The `<icesub:delivery-rule` element is defined as the type *delivery-ruleType* and is described by this XML Schema fragment:

```
<xs:complexType name = "delivery-ruleType">
  <xs:sequence>
      <xs:element name = "transport"
          maxOccurs = "unbounded" minOccurs = "1"
          type = "transportType"/>
    <xs:any namespace = "##local ##other"
        processContents = "lax" minOccurs = "0"
        maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:attribute name = "mode" default = "pull">
   <xs:simpleType>
    <xs:restriction base = "xs:NMTOKEN">
      <xs:enumeration value = "pull"/>
      <xs:enumeration value = "push"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name = "monthday" type = "xs:NMTOKENS"/>
  <xs:attribute name = "weekday" type = "xs:NMTOKENS"/>
  <xs:attribute name = "startdate" type = "icesdt:dateTime"/>
  <xs:attribute name = "stopdate" type = "icesdt:dateTime"/>
  <xs:attribute name = "starttime" type = "icesdt:time"/>
  <xs:attribute name = "duration" type = "icesdt:duration"/>
  <xs:attribute name = "min-num-updates"
        type = "xs:integer"/>
  <xs:attribute name = "max-num-updates"
        type = "xs:integer"/>
  <xs:attribute name = "incremental-update"
        type = "xs:boolean" default = "false"/>
  <xs:attribute name = "required" type = "xs:boolean"
        default = "true"/>
  <xs:anyAttribute namespace = "##other"
        processContents = "lax"/>
</xs:complexType>
```

Attributes on `<icesub:delivery-rule` include:

- **mode**
  Default. This attribute specifies the mode for the delivery. Options are `push` from Syndicator to Subscriber and `pull` by Subscriber from Syndicator, with a default of "`pull`" to support Basic ICE.
- **incremental-update**
  Default. This attribute specifies the update policy for the offer. The values are Boolean with "false" as the update default.
- **required**
  Default. This attribute specifies whether this delivery rule is required in order for the offer to be accepted. The values are Boolean with the default as "true".
- **startdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the date and time on which the delivery schedule will start to apply. If this attribute is omitted, the schedule will start immediately.
- **stopdate**
  Optional. Datatype is `icesdt:dateTime` as defined in the ICE datatype schema. This attribute specifies the date and time on which the delivery schedule expires. If this attribute is omitted, the schedule never expires (unless superseded in the future).
- **weekday**
  Optional. This token indicates the day of the week on which delivery is scheduled .
- **monthday**
  Optional. This token indicates the day of the month on which delivery is scheduled.
- **duration**
  Optional. Datatype `icesdt:duration` as defined in the ICE datatype schema. This attribute specifies the duration of the window that starts at start-time everyday. .
- **min-num-updates**
  Optional. This attribute specifies the minimum number of updates. The datatype is an integer.
- **max-num-updates**
  Optional. This attribute specifies the maximum number of updates. The datatype is an integer.

5.4.5.4.1.1 Transport

The `<icesub:delivery-rule` is made up of one or more `<icesub:transport.` Transports are specified when the Syndicator makes an offer. This element provides a mechanism for the Syndicator to indicate the possible delivery transports for the `<icesub:offer`. You can see the makeup of a `<icesub:transport` in the Figure 5.9:

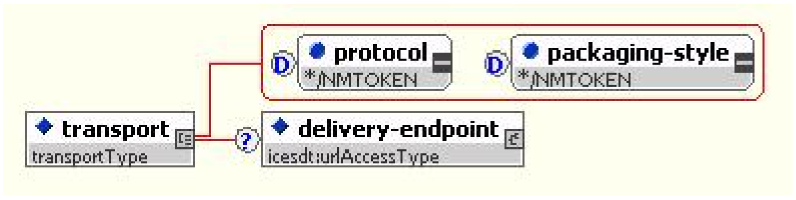


Figure 5.9 Transport structure

The <`icesub: transport has two attributes:`

- **protocol**
  Default. This attribute specifies the transport protocol. It has pre-enumerated values of "`http:get`" "`ftp`" "`mailto`" and "`soap`" with the default set to "`http:get`" for Basic ICE.
- **packaging-style**
  Default. This attribute specifies the packaging style for the offer. It has pre-enumerated values of "`ice`" and "`raw`" with the default set to "`ice`" for Basic ICE.

5.4.5.4.1.2 Delivery-Endpoint

The <icesub:transport is made up of an optional delivery endpoint. See Figure 5.10. If the Syndicator is offering content in "`pull`" mode, the delivery endpoint can be specified by using `<icesub:delivery-endpoint`. If the Syndicator is offering content in "`push`" mode, the Subscriber would use this elements within the `<icesub:subscribe` message to indicate the endpoint for the push delivery.

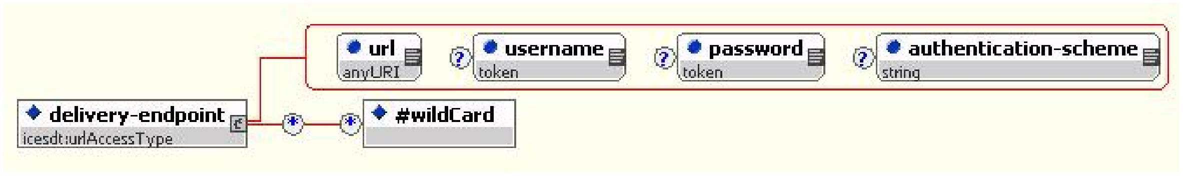


Figure 5.10 Delivery-endpoint structure

The `<icesub:delivery-endpoint` has 4 attributes. These include:

- **url**
  Required. This attribute specifies the URL for push delivery. The datatype is `anyURI.`
- **username**
  Optional. This attribute specifies an optional username that may be required to access URL for push delivery.
- **password**
  Optional. This attribute specifies an optional password that may be required to access URL for push delivery.
- **user-authentication**
  Optional. This attribute specifies the optional user authentication scheme. It is a string with enumerated values of "`basic`" and "`digest`". There is no default

5.4.5.4.2 Syndicator Offer Specifications by Mode

One of the most important specifications within the delivery rule of an offer is the specification of delivery `mode=` along with the `<icesub:syndicator-transports`. The requirement to specify syndicator transports delivery settings varies by delivery mode. Conditions such as this cannot be expressed by XSD. The following table provides required specifications based on delivery mode.

Mode	Syndicator Protocol	Syndicator Delivery Packaging Style	Delivery Endpoint
Pull	**Default**. If not specified, the default protocol will be "http:get"	**Default**. If not specified, Syndicator packaging is assumed to be "ice"	**Optional.** If not specified it is assumed to be the same endpoint from where the catalog was pulled
Push	**Required**. For push delivery a specific protocol should be indicated because http:get is not a push protocol	**Required**. For push delivery a specific protocol should be indicated	**Not Allowed.** For push delivery, only the Subscriber delivery endpoint is valid

5.4.5.4.3 Example Delivery Rules

In this section we will look at a number of offers with delivery rules within delivery policies. The intent is to provide examples of delivery rules with different modes and Syndicator specifications.

5.4.5.4.2.1 Simple "Pull" Delivery Rule

First lets look at an offer with a `<icesub:delivery-rule` containing a simple "`pull`". Notice that in this simple rule, everything is left to default including the mode on the delivery rule. No transport protocol or packaging-style is provided. Remember that the assumption is that the protocol will be "`http:get`" and the packaging will be "`ice`". The pull will be made from the same location from which the catalog was pulled.

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
    <icedel:item>
      <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
       <icesub:delivery-policy>
          <icesub:delivery-rule/>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

NOTE: You can tell this ICE package contains a catalog offer in several ways. First notice that the `subscription-id` on the package equals "`1`". This is the identifier of a subscription catalog. Also notice that the `<icedel:metadata` indicates the item type is offer. And finally the offer is inside this package.

5.4.5.4.2.2 “Pull” Delivery Rule with Syndicator Delivery Settings

In this example, the delivery rule specifies a pull delivery. But rather than using the defaults, this Syndicator is specifying transport. In this case the Syndicator provides a delivery endpoint for content to be pulled from. The Syndicator also indicates that the delivery packaging style will “`ice`”.

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
    <icedel:item>
      <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
       <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
          <icesub:delivery-rule mode="pull"/>
              <icesub:transport
                protocol=”http:get” packaging-style=”ice”>
                <icesub:delivery-endpoint
                  url="http://iceserver.com/gp/08292BC"/>
              </icesub:transport>
            </icesub:delivery-rule>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

NOTE: The `<icesub:delivery-policy` indicates that this subscription provides for a quantity of 100 feeds. In this case, the subscription expires when the quantity has been filled. Also note that no times or durations are placed on this subscription.

5.4.5.4.2.3 Single "Push" Delivery Rule

Instead of specifying that delivery will be by "`pull`", the Syndicator may indicate a "`push`" delivery. In this case the Syndicator provides protocol and packaging information but does not provide `<icesub:delivery-endpoint` as push endpoints have to be provided by the Subscriber!

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
    <icedel:item>
      <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
       <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
         <icesub:delivery-rule mode="push">
          <icesub: transport protocol="soap"
              packaging-style="ice"/>
          <icesub:transport protocol="soap"
              packaging-style="raw"/>
          <icesub:transport protocol="ftp"
              packaging-style="ice"/>
          <icesub:transport protocol="ftp"
              packaging-style="raw"/>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

NOTE: In this example the Syndicator provided four transport protocols/packaging style options. The Subscriber can select a preferred transport protocol and packaging style pair when subscribing to this offer.

5.4.5.4.2.4 Combined “Pull” and “Push” Delivery Rule

A Syndicator may specify that delivery will be by “`pull`” and “`push`” delivery. In this case the Subscriber must be able to accept both delivery rules (which default to required) in order to subscribe to the offer. The Subscriber can, however, choose a preferred transport within each rule.

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
   <icedel:item>
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
      <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
          <icesub:delivery-rule mode="pull">
           <icesub:transport protocol=”http:get”
              packaging-style=”ice”>
             <icesub:delivery-endpoint
            url="http://iceserver.com/ice/08292BC82302427"/>
           </icesub:transport>
           <icesub:transport protocol=”http:get”
              packaging-style=”raw”>
             <icesub:delivery-endpoint
            url="http://iceserver.com/raw/08292BC82302427"/>
           </icesub:transport>
         </icesub:delivery-rule>
         <icesub:delivery-rule mode="push">
          <icesub:transport>
           <icesub:transport protocol=”soap”
              packaging-style=”ice”/>
           <icesub:transport protocol=”mailto”
              packaging-style=”ice”/>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
      </icesub:offer>
    </icedel:item>
  </icedel:add>
</icedel:package>
```

NOTE: Because the “`required`” attribute on delivery rule is left to default to “`true`” all delivery rules within this delivery policy must be accepted in order for the Subscriber to subscribe to the offer.

5.4.5.5 Offer Business Term

Another component of the `<icesub:offer` is the optional `<icesub:business-term` element. Business terms provide the means for additional content and parameters to be communicated between the parties; both for specific subscriptions as well as for more general properties of the relationship. You can see the structure of this element in Figure 5.11:

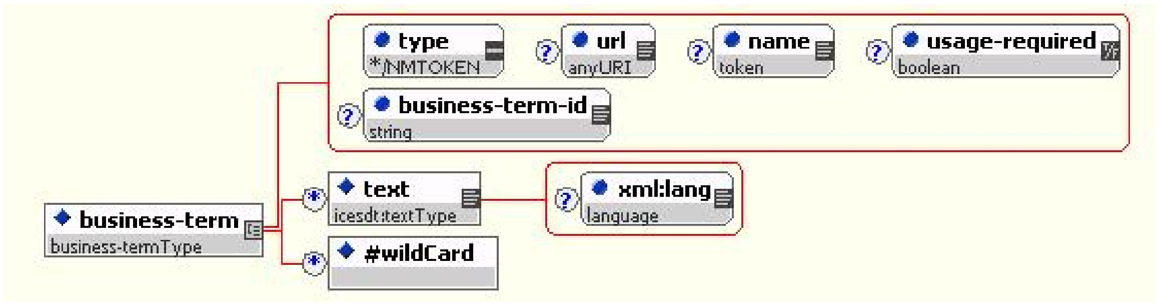


Figure 5.11 ICE business-term structure

The `<icesub:business-term` element is defined as the type *business-termType* and is described by this XML Schema fragment:

```
<xs:element name = "business-term" type = "business-termType"
minOccurs = "0" maxOccurs = "unbounded"/>
  <xs:complexType name = "business-termType" mixed = "true">
   <xs:sequence>
    <xs:element name = "text" type = "icesdt:textType"
minOccurs = "0" maxOccurs = "unbounded"/>
    <xs:any namespace = "##other" processContents = "lax"
minOccurs = "0" maxOccurs = "unbounded"/>
   </xs:sequence>
   <xs:attribute name = "type" use = "required">
    <xs:simpleType>
     <xs:restriction base = "xs:NMTOKEN">
     <xs:enumeration value = "credit"/>
     <xs:enumeration value = "licensing"/>
     <xs:enumeration value = "payment"/>
     <xs:enumeration value = "reporting"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name = "url" type = "xs:anyURI"/>
  <xs:attribute name = "name" type = "xs:token"/>
  <xs:attribute name = "usage-required" type = "xs:boolean"/>
  <xs:attribute name = "business-term-id" type =
"xs:string"/>
  <xs:anyAttribute namespace = "##other" processContents =
"lax"/>
```

```
</xs:complexType>
```

The attributes of `<icesub:business-term` are:

- **type**
  Required. String identifying the particular class of business terms. All of these terms are plain text descriptions. ICE makes no attempt to programmatically explain licensing agreements; rather, ICE simply provides a transport mechanism allowing user interfaces to easily locate, manage, and display electronic copies of license agreements presumably executed in the traditional way on paper. The type is one of the following values:

type attribute value	**Attribute "type" Meaning**
credit	Refers to the type of acknowledgement required when using the content. Note that ICE makes no further requirements about credit. The party making the offer (usually a syndicator) **MAY** choose to provide parameters in this category that MAY or MAY NOT be negotiable.
licensing	Refers to the general terms of licensing. Note that ICE makes no further requirements about licensing. The party making the offer (usually a syndicator) **MAY** choose to provide parameters in this category that MAY or MAY NOT be negotiable.
payment	Payment refers to the cost and payment terms expected when using the content. Note that ICE makes no further requirements about payment. The party making the offer (usually a syndicator) **MAY** choose to provide parameters in this category that MAY or MAY NOT be negotiable.
reporting	Refers to the end-user usage statistics expected when content is used. Note that ICE makes no further requirements about reporting (but see logging). The party making the offer (usually a syndicator) **MAY** choose to provide parameters in this category that MAY or MAY NOT be negotiable.

- **url**
  Optional. A url. URL has no protocol-defined semantics other than to be made available to the ICE application processor. The intent is that this URL provides the business terms.
- **name**
  Optional. This name MAY be used by an ICE application processor to identify the specific business term. "name" has no protocol defined semantics other than to be made available to the ICE application processor.
- **usage-required**
  Optional. This attribute specifies whether the business term usage is required. This

is a Boolean. If `true`, indicates usage is required and `false` indicates it is not required.

- **business-term-id**
  Optional. A subscription unique business term identifier that ICE uses to distinguish the business term from all other business terms in the subscription.

5.4.5.6 Offers for RSS Feeds

ICE 2.0 has specifically been designed to carry “`rss`” feeds. RSS is a simple mechanism for enabling the lightweight syndication of content. RSS was designed to be simple to use and inexpensive to implement. RSS has been widely deployed, but remains limited in its ability to enforce business rules in the content syndication environment or to push content to Subscribers. ICE 2.0 was designed to add these capabilities to RSS in an automated Web Services environment.

In order to use ICE 2.0 to carry RSS feeds, one key element of an ICE offer is used:

- The `item-type=` attribute on the `<icesub:content-metadata` of an ICE offer was designed to enable the specification which version of RSS would be used in the feed. Currently RSS versions (developed and managed by different organizations) include RSS0.91, RSS0.92, RSS1.0 and RSS2.0. The `item-type=` attribute has anyURI as a value so it can point to other specifications as well.

The following example shows an ICE offer for an RSS feed:

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="1"
     subscription-id="1">
  <icedel:add>
   <icedel:metadata item-
type="http://icestandard.org/ICE/V20/item-type/offer"
content-type="text/xml"/>
   <icedel:item>
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID4"
    name="offName4">
<icesub:content-metadata
      atomic-use="true"
      editable="false"
      ip-status="Free With Acknowledgement"
      rights-holder="Oracle Corporation, 2003"
      show-credit="true"
      item-type="http://icestandard.org/ICE/V20/item-
type/rss2.0"/>
      <icesub:description>
           headlines in RSS
      </icesub:description>
      <icesub:delivery-policy quantity="100"
```

```
          expiration-priority="quantity">
           <icesub:delivery-rule mode="push">
            <icesub: transport protocol="soap"
              packaging-style=”ice”/>
            <icesub:transport protocol=”soap”
              packaging-style=”raw”/>
           </icesub:delivery-rule>
         </icesub:delivery-policy>
        </icesub:offer>
      </icedel:item>
    </icedel:add>
</icedel:package>
```

NOTE: This ICE offer adds push functionality to an RSS feed. It also provides business rules that limit the quantity of the feed.

5.4.6 Subscribing

A Subscriber uses the `<icesub:subscribe` containing an `<icesub:offer` to establish a subscription. Typically, a Subscriber will use `<icedel:get-packages/<icedel:get-package` to get an initial package of offers, take one of the offers from that catalog of offers and send it back to the Syndicator in an `<icesub:subscribe` request.

5.4.6.1 Subscribe Element

The subscribe message is made up of an offer with parameters. See Figure 5.12.

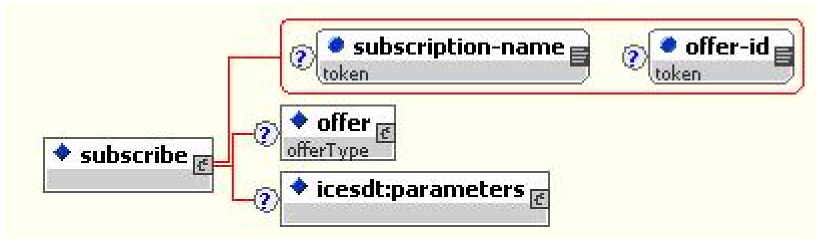


Figure 5.12 The structure of the subscribe message

The `<icesub:subscribe` message can carry only a single offer. This means that there is a single offer per subscription. The `<icesub:offer` element is described in *5.4.3 Offers*.

In addition to the offer, the `<icesub:subscribe` message may contain the `<icesub:parameters` element. This element enables the Subscriber to send further parameters at subscription time to specify parameters. These parameters are not defined within an ICE 2.0 namespace, but rather must come from a Subscriber namespace.

The structure of the `<icesub:subscribe` message is shown in the XML schema fragment:

```
<xs:element name = "subscribe">
  <xs:complexType>
    <xs:sequence>
     <xs:element name = "offer" type = "offerType" minOccurs
= "0"/>
     <xs:element ref = "icesdt:parameters" minOccurs = "0"/>
    </xs:sequence>
    <xs:attribute name = "subscription-name" type =
"xs:token"/>
    <xs:attribute name = "offer-id" type = "xs:token"/>
    <xs:anyAttribute namespace = "##other" processContents =
"lax"/>
  </xs:complexType>
</xs:element>
```

The `<icesub:subscribe` message has a two attributes:

- **subscription-name**
  Optional. This attribute specifies the name of the product being subscribed to.
- **offer-id**
  Optional. This attribute specifies the id of the offer being subscribed to. If this attribute is used, without an echo of the `<icesub:offer`, it means that the offer was accepted just as it was presented.

5.4.6.1.1 Subscribing Directly to an Offer

If an `<icesub:subscribe` is returned with the `offer-id` attribute but without an echo of the `<icesub:offer`, it means that the offer was accepted and subscribed to just as it was presented. The `offer-id` attribute was put on `<icesub:subscribe` specifically to allow for this short cut.

See how this is done in the following example:

```
<icesub:subscribe subscription-name="RSS Headlines"
     offer-id="offID2"/>
```

NOTE: The `offer-id` can only be used to subscribe to offers that are “pull” only. If an offer has “push” delivery rules, the Subscriber must return the offer with delivery endpoints for the push specified.

5.4.6.1.2 Subscribing with Subscriber Parameters Returned

If an offer has “push” delivery rules, the Subscriber must return the offer with delivery endpoints for the push specified. The Subscriber may also have been given choices of delivery style selections that must be specified in order for content delivery to commence.

In both these cases, the Subscriber must return the `<icesub:offer` within the `<icesub:subscribe`.

5.4.6.1.2.1 Subscriber Transport

The Subscriber returns transport for push deliveries back to the Syndicator within the offer that is returned in the `<icesub:subscribe` message. For information on `<icesub:transport`.

5.4.6.1.2.2 Example Subscribe Message with Subscriber Parameters

This example shows an `<icesub:subscribe` message in response to the offer shown in *5.4.5.4.2.4 Combined "Pull" and "Push" Delivery Rule.* In this example, the Subscriber sends the offer within the `<icesub:subscribe` message. Note that the Subscriber has selected one `<icesub:transport` option. Also `<icesub:delivery-endpoint` has been provided by the Subscriber so the Syndicator will know where the push delivery will be made.

```
<icesub:subscribe>
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
      <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
         <icesub:delivery-rule mode="push">
          <icesub:transport protocol="soap"
              packaging-style=”ice”>
            <icesub:delivery-endpoint
              url="http://sub.com/push.jsp" username="foo"
              password="foofoo"/>
           </icesub:transport>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
    </icesub:offer>
</icesub:subscribe>
```

5.4.6.2 Subscription Initiated

After the Subscriber returns the offer to the Syndicator within a `<icesub:subscribe` message, the Syndicator can respond in one of two ways, depending upon whether the subscription was accepted.

If the Syndicator accepts the subscribe request, the Syndicator responds with the `<icesub:subscription` message shown in Figure 5.13.

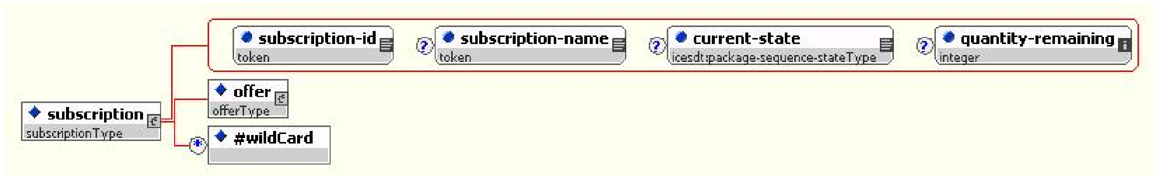


Figure 5.13 Syndicator's Subscription Response

The structure of the subscription message can be seen in this XML schema fragment:

```
<xs:element name = "subscription" type =
     "subscriptionType"/>
  <xs:complexType name = "subscriptionType">
   <xs:sequence>
    <xs:element name = "offer" type = "offerType"/>
     <xs:any namespace = "##other" processContents = "lax"
minOccurs = "0" maxOccurs = "unbounded"/>
   </xs:sequence>
   <xs:attribute name = "subscription-id" use = "required"
type = "xs:token"/>
   <xs:attribute name = "subscription-name" type =
"xs:token"/>
   <xs:attribute name = "current-state" type =
"icesdt:package-sequence-stateType"/>
   <xs:attribute name = "quantity-remaining" type =
"xs:integer"/>
   <xs:anyAttribute namespace = "##other" processContents =
"lax"/>
</xs:complexType>
```

The `<icesub:subscription` element has several attributes:

- **subscription-id**
  Required. This attribute specifies the unique identifier of the product being subscribed to. The Syndicator provides a subscription-id when the subscription begins.
- **subscription-name**
  Optional. This attribute specifies the name of the product being subscribed to.
- **current-state**
  Optional. This attribute specifies the current state of the subscription. It is datatype `icesdt:package-sequence-stateType` as defined in the ICE simpledatatypes.xsd. Values include `ICE-INITIAL` and `ICE-ANY`.
- **quantity-remaining**
  Optional. This attribute specifies the quantity of updates of the product being subscribed to. The datatype is an integer.

NOTE: The subscription does not have an offer-id as an attribute. This means that even though the offer-id can be used as a short-cut by the Subscriber when subscribing, the Syndicator is forced to repeat the entire offer within the subscription. This is a safeguard to ensure that the Subscriber clearly understands the subscription and all delivery policies at the time the subscription in initiated by the Syndicator.

An example of the `<icesub:subscription` message is shown below. This is the subscription that was established based on the example shown in *5.4.6.1.2.2 Example Subscribe Message with Subscriber Parameters.*

```
<icesub:subscription
       subscription-id="08292BC82302427F8CBC93342F931EC8"
       current-state="ICE-INITIAL"
       quantity-remaining="100">
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2" name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
      <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
         <icesub:delivery-rule mode="push">
          <icesub:transport protocol="soap"
              packaging-style=”ice”>
            <icesub:delivery-endpoint
              url="http://sub.com/push.jsp" username="foo"
              password="foofoo"/>
           </icesub:transport>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
    </icesub:offer>
</icesub:subscription>
```

Note The subscription message is returned directly within the SOAP body. Even though the original offer is sent inside a package, the subscription reply is not.

5.4.6.3 Subscription Declined

If the Syndicator declines the subscription, the response is `<icesub:subscription-fault`. The fault contains a fault code. One of the following fault codes is appropriate for declining a subscription.

- 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.*
- 422 Schedule violation
  The subscribe request was made at an incorrect time such as after an offer has expired or before it is valid.
- 440 Sorry
  This indicates the Syndicator rejected the proposed subscription offer, but wishes to extend additional offers.

5.4.6.4 ICE Subscription Fault

The `<icesub:subscription-fault` is returned when a subscription is declined. The structure of the fault can be seen in Figure 5.14.

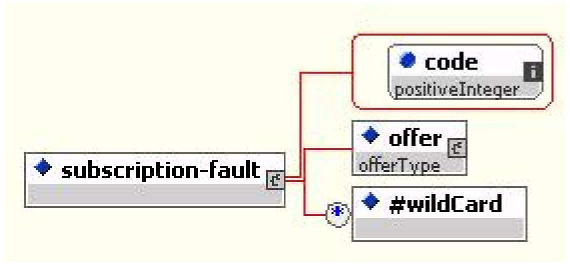


Figure 5.14 ICE subscription fault

The following is an example of a Syndicator declining a subscription:

```
<icesub:subscribe-fault
           code="440">
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2" name="offName2">
      <icesub:description>
           headlines
      </icesub:description>
      <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
         <icesub:delivery-rule mode="push">
          <icesub: transport protocol="soap"
              packaging-style=”ice”>
            <icesub:delivery-endpoint
              url="http://sub.com/push.jsp" username="foo"
              password="foofoo"/>
           </icesub:transport>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
    </icesub:offer>
</icesub:subscription>
</icesub:subscribe-fault>
```

5.5 Other Subscription Operations

In addition to providing the ability for the Subscriber to subscribe to an offer and for the Syndicator to approve and manage that subscription, Full ICE provides for two other important subscription management operations— checking the status of a subscription and cancellation of the subscription.

5.5.1 Get Status

ICE 2.0 provides the ability for the Subscriber to request the status of a subscription. The structure of the `<icesub:get-status` message is shown in Figure 5.15.

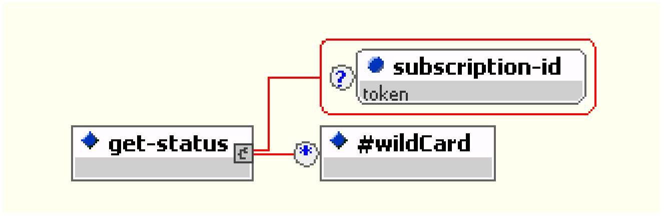


Figure 5.15 Get-status Structure

NOTE: If the optional subscription-id is not provided, the Syndicator is expected to respond with the status of each subscription for the Subscriber.

The following example shows how the <icesub:get-status request is issued. Note that the entire ICE/SOAP message is shown.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/12/soap-
envelope'>
  <env:Header>
    <icemes:Header timestamp="2003-03-03" message-id="m0056">
      <icemes:Sender name="mycompany"
         role="http://icestandard.org//role/syndicator"
         sender-id="http://www.xxyz.org"/>
      </icemes:Header>
  </env:Header>
  <env:Body>
    <icesub:get-status subscription-id="MC003"/>
  </env:Body>
<env:Envelope>
```

5.5.2 Status

ICE 2.0 provides the ability for the Syndicator to respond to the request from the Subscriber for the status of a subscription. The structure of the <icesub:status message is shown in Figure 5.16.

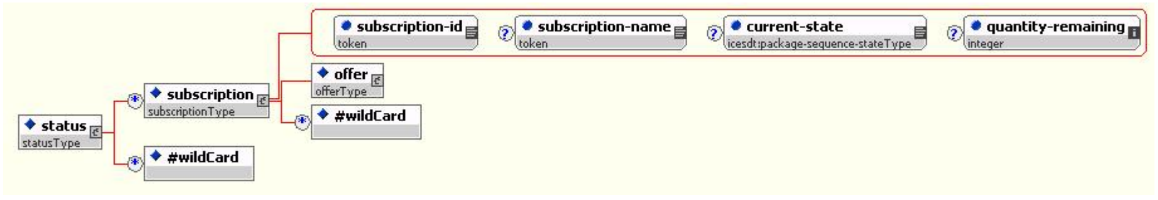


Figure 5.16 ICE Status Structure

The ICE Status response returns the subscription element that includes the current state of the subscription, the quantity remaining in the subscription and the subscription-id along with the offer itself. From this information, the Subscriber can answer any question that prompted the <icesub:get-status request.

5.5.3 Cancel

ICE 2.0 provides the ability for the Subscriber to cancel a subscription. The structure of the <icesub:cancel message is shown in Figure 5.17.

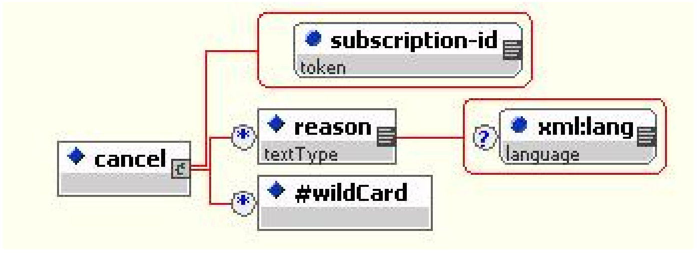


Figure 5.17 ICE Cancel Structure

The Subscriber’s request to cancel a subscription simply includes the subscription-id for the subscription being cancelled and a reason attribute. The `xml:lang` attribute enables the Subscriber to specify the language for the reason text.

An example of an ICE cancel message is shown here:

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/12/soap-
envelope'>
  <env:Header>
    <icemes:Header timestamp=”2003-03-03” message-id=”m0056”>
      <icemes:Sender name=”mycompany”
         role=”http://icestandard.org//role/syndicator”
         sender-id=”http://www.xxyz.org”/>
      </icemes:Header>
  </env:Header>
  <env:Body>
<icesub:cancel
      xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
      subscription-id="08292BC82302427F8CBC93342F931EC8">
  <icesub:reason xml:lang="en">
    I'm tired of this content feed
  </icesub:reason>
</icesub:cancel>
</env:Body>
</env:Envelope>
```

5.5.4 Cancellation

ICE 2.0 provides the ability for the Syndicator to verify the cancellation of a subscription requested by the Subscriber with the `<icesub:cancel` message. The structure for the `<icesub:cancellation` response is shown in Figure 5.18.

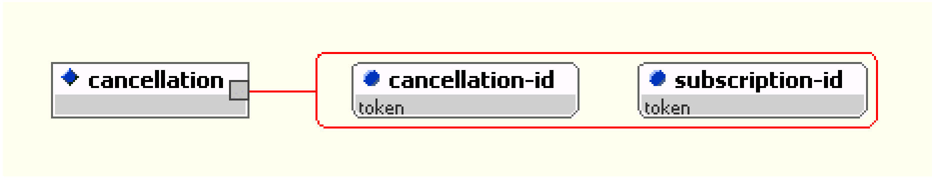


Figure 5.18 ICE Cancellation Response Structure

The Cancellation response requires the Syndicator to provide the Subscriber with a unique cancellation-id that can be used to verify the cancellation.

An example of an ICE cancellation response is shown here:

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/12/soap-
envelope'>
  <env:Header>
    <icemes:Header timestamp=”2003-03-03” message-id=”m0056”>
      <icemes:Sender name=”mycompany”
         role=”http://icestandard.org//role/syndicator”
         sender-id=”http://www.xxyz.org”/>
      </icemes:Header>
  </env:Header>
  <env:Body>
<icesub:cancellation
xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
        subscription-id="08292BC82302427F8CBC93342F931EC8"
        cancellation-id="C08292BC82302427F8CBC93342F931EC8"/>
</env:Body>
</env:Envelope>
```

5.6 Packages and Delivery

Full ICE, like Basic ICE, supports the delivery of packages. In Basic ICE, the delivery is simply the act of the Syndicator placing the content in a SOAP/ICE XML document at the URL specified in the offer. Full ICE enables the push or pull of content. The XML definition for packages is found at http://www.icestandard.org/Spec/V20/schema/ice-delivery.xsd.

The Full ICE package is made up of three elements. See Figure 5.19

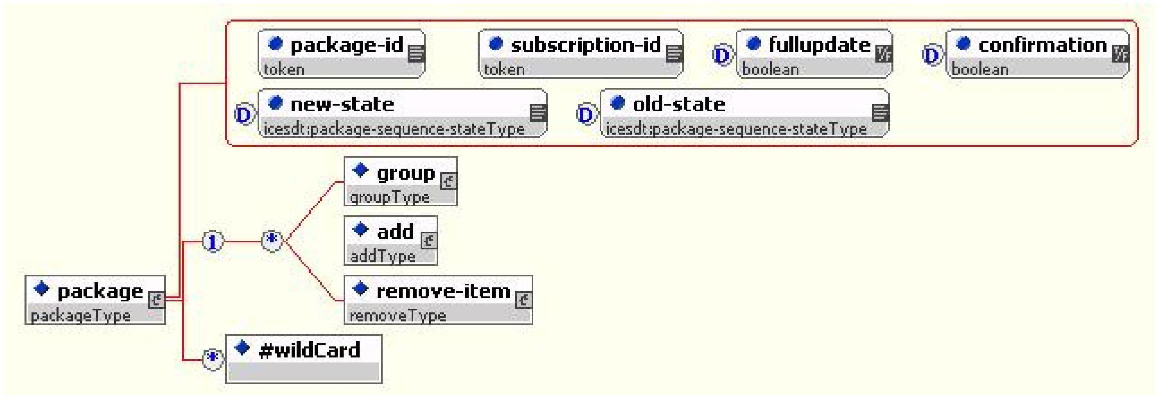


Figure 5.19 ICE package structure

The formal definition of a package is expressed with the following XML schema. Note the defaults of package attributes that define Basic ICE functionality.

```
<element name = "package" type = "icedel:packageType"/>
  <complexType name = "packageType">
    <sequence>
      <group ref = "icedel:cm.package"/>
```

```
        <any namespace = "##other" processContents = "lax"
minOccurs = "0"
             maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "package-id" use = "required" type =
"token"/>
    <attribute name = "subscription-id" use = "required" type
= "token"/>
    <attribute name = "fullupdate" default = "true" type =
"boolean"/>
    <attribute name = "confirmation" default = "false" type =
"boolean"/>
    <attribute name = "new-state" default = "ICE-ANY"
        type = "icesdt:package-sequence-stateType"/>
     <attribute name = "old-state" default = "ICE-ANY"
        type = "icesdt:package-sequence-stateType"/>
    <anyAttribute namespace = "##other" processContents =
"lax"/>
  </complexType>
</element
```

5.6.1 Package Attributes

There are several attributes on `package`:

- **confirmation**
  Default. This attribute specifies whether confirmation of receipt is required. The values are Boolean, “true” or “false.” The default is “false”. Confirmation does not apply for Basic ICE since Basic ICE does not support subscription management.
- **fullupdate**
  Default. This attribute specifies whether the package contains a full (or partial) update. The values are Boolean, “true” or “false.” The default is “true” because Basic ICE does not require management of incremental updates.
- **new-state**
  Default. One of two sequence identifiers, which, together represent the state of the subscription. Since Basic ICE does not support subscription management, the default is set to “ICE-ANY”.
- **old-state**
  Default. One of two sequence identifiers, which, together represent the state of the subscription. Since Basic ICE does not support subscription management, the default is set to “ICE-ANY”.
- **package-id**
  Required. Identifies the package within the scope of a subscription. It is referenced in certain `ice-code` messages such as 201 (Confirmed) and for package confirmations. The Syndicator assigns the `package-id`.

- **subscription-id**
  Required. In Basic ICE, the subscription-id is the unique id of the content feed and is used by all subscribers. The Syndicator assigns the `subscription-id`.

5.6.2 Package Elements

The ICE package is made up of 3 elements. See figure 5.20. An ICE `<icedel:package` describes a set of content operations: additions, removals, and a group of additions and/or removals that are used to update/distribute syndicated content. The content additions contain the content that needs to be added or updated and are specified using the `<icedel:item` and `<icedel:item-ref` elements. The `<icedel:group` element allows the Syndicator to associate the content specified using the `<icedel:item` elements together. For example, in the syndication of restaurant reviews, each review may consist of different types of content such as an HTML file and two graphic files. These three files could be contained within three `<icedel:item` elements and grouped together in an ICE `<icedel:group` as a single restaurant review. Likewise, unrelated content can be specified in a `<icedel:package` by just using the `add and then <icedel:item` elements without an intervening `<icedel:group`. The `<icedel:item` element is used to contain content directly for delivery. The `<icedel:item-ref` element is used to distribute an indirect reference to the actual content. Note that the `#wildcard` allows for insertion of content from any namespace.

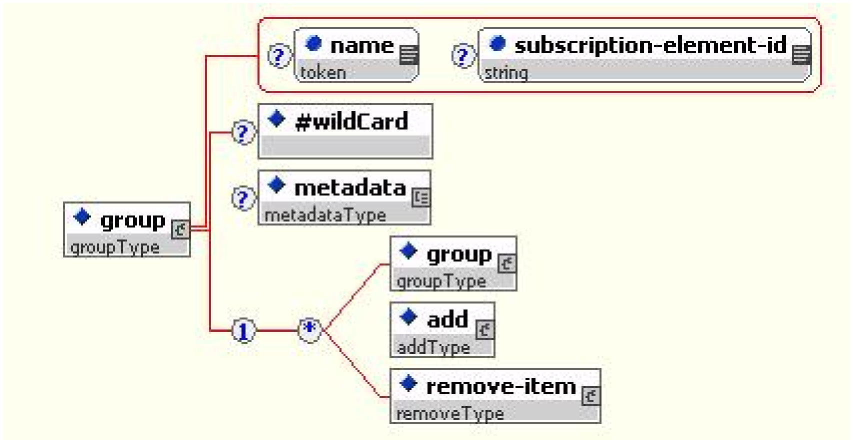


Figure 5.20 Package elements for Full ICE

5.6.2.1 Group

The `<icedel:group` is a container element that can be used to group content items being added or removed. It also enables the attachment of metadata to a group of content items.

Attributes on `<icedel:group` include:

- **name**
  Optional. This attribute specifies a name for the item group that can be uwsed to identify that group within a package.
- **subscription-element-id**
  Optional. This attribute specifies the persistent identifier of thegroup of elements within the subscription

5.6.2.2 Metadata

The `<icedel:metadata` element enables the entry of metadata on `<icedel:group`, `<icedel:add`, or `<icedel:remove-item` by using its attributes and description element.

Attributes on `<icedel:metadata` are shown in Figure 5.21.

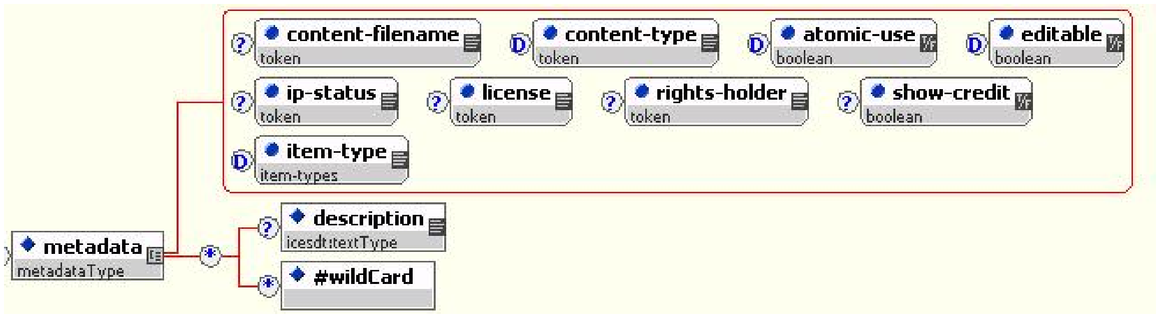


Figure 5.21 Attributes on Metadata element

- **content-filename**
  Optional. This element enables the specification the file name contained within.
- **content-type**
  Optional. This attribute enables the specification of the type of content such as “news.”
- **atomic-use**
  Optional. This attribute specifies whether the element may be used in part. The values are Boolean, “true” or “false.” The default is “false” because Basic ICE requires complete item usage.
- **editable**
  Optional. This attribute specifies whether the element is editable or whether it must be used as delivered. The values are Boolean, “true” or “false.” The default is “false” because Basic ICE requires unaltered item usage.
- **ip-status**
  Optional. This specifies the intellectual property right status. The value is a token.
- **license**
  Optional. This specifies the license status of the content. The value is a token.

- **rights-holder**
  Optional. This attribute specifies the rights holder. The value is a token
- **show-credit**
  Optional. This attribute specifies the requirement to show credit for the content. The values are Boolean, “true” or “false.” There is no default.

- **item-type**
  Optional. This attribute specifies a URI that identifies what type of item this is. For example the value of item-type may be "`http://icestandard.org/ICE/V20/item-type/offer`".

5.6.2.3 Add

The `<icedel:add` element is used to add new content according to the delivery policy of the subscription. It enables the attachment of metadata to the content being added. The `<icedel:add` enables content to be directly included in the message by using the `<icedel:item` element, an indirect reference to content using `<icedel:item-ref` mechanism.

The structure of `<icedel:add` is shown in Figure 5.22.

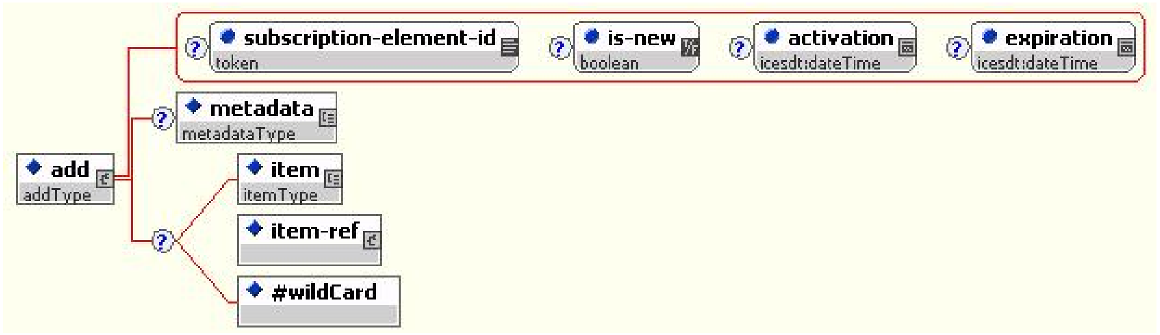


Figure 5.22 Add element structure

The `<icedel:add` element includes the following attributes:

- **subscription-element-id**
  Optional. This attribute specifies the persistent identifier of an element of a subscription that is being added. This applies to the contained item, item-ref or syndicator supplied (`#wildcard`) content.
- **is-new**
  Optional. This attribute specifies that the content is new. The values are Boolean, "`true`" or "`false`." There is no default.
- **activation**
  Optional. This attribute specifies when the addition for content is activated. The value is in the `icesdt:dateTime` format.
- **expiration**
  Optional. This attribute specifies when the content expires. This attribute specifies when the addition for content is activated. The value is in the `icesdt:dateTime` format.

5.6.2.4 Remove Item

The `<icedel:remove-item` element is used to remove content of the subscription.

The structure of `<icedel:remove-item` is shown in Figure 5.23.

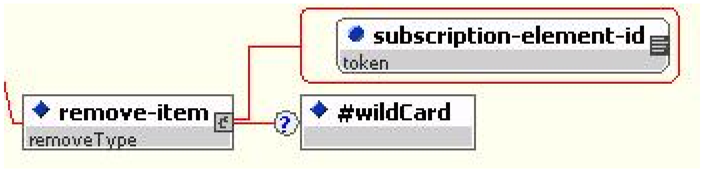


Figure 5.23 Remove item structure

The `<icedel:remove-item` element has a single required attribute that identifies what is to be removed:

- **subscription-element-id**
  Required. This attribute specifies the persistent identifier of an element of a subscription

5.6.2.5 Item

The `<icedel:item` element directly carries content from the Syndicator to the Subscriber. An `<icedel:item` can carry the ICE `<icesub:offer`. The `<icedel:item` does not carry subscription management elements such as `<icesub:subscribe` or `<icesub:cancel`. The `<icedel:item` structure is shown in Figure 5.24.

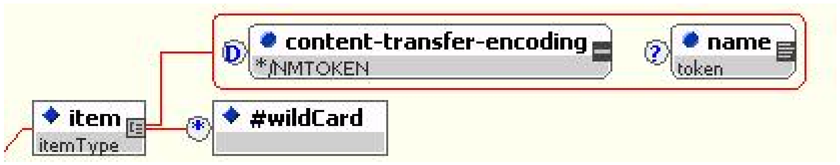


Figure 5.24 Item structure

The `<icedel:item` has two attributes:

- **content-transfer-encoding**
  Default. This attribute specifies the transfer encoding. Choices are `base64` or `x-native-xml` with `x-native-xml` as the default.
- **name**
  Optional. This attribute specifies the item name that can be used as a transient identifier within a group or add.

5.6.2.6 Item-Ref

The `<icedel:item-ref` element references Syndicator content. The `<icedel:item-ref` structure is shown in Figure 5.25. It is made up of a single `<icedel:reference` element. This means that for each reference, an `<item-ref` element must be used.

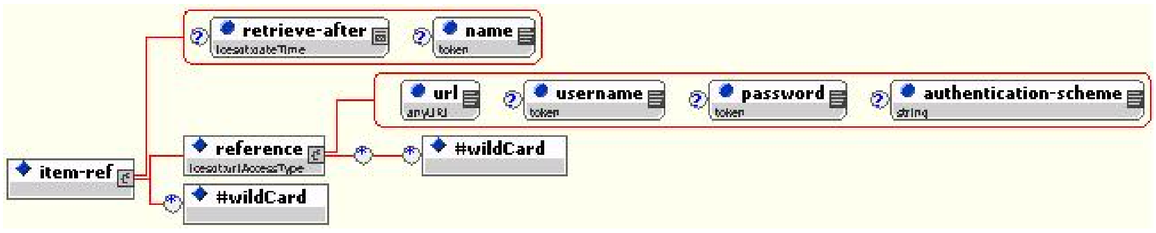


Figure 5.25 Item-ref structure

The `<item-ref` element has two attributes:

- **retrieve-after**
  Optional. This attribute specifies a time after which the item can be retrieved. It is specified in the `icesdt:dateTime` format.
- **name**
  Optional. This attribute specifies the item name that can be used as a transient identifier within a group or add.

5.6.2.7 Reference

The `<icedel:reference` element is used to reference the content of the `<icedel:item-ref` element. The reference element is empty (with the exception of any wildcard content). The attributes carry the information for this element.

The `<icedel:reference` element has four attributes:

- **url**
  Required. This attribute specifies the URL from which the content can be retrieved.
- **username**
  Optional. This attribute specifies the username for retrieving the content if a login is required.
- **password**
  Optional. This attribute specifies the password for retrieving the content if a login is required.
- **authentificationscheme**
  Optional. This attribute specifies the authentification scheme for retrieving the content if this is required.

5.6.3 Package Confirmations

Full ICE has a mechanism to confirm the delivery of packages. If the package that is being delivered has `confirmation="yes"` then the Full ICE Subscriber must return a package confirmations response. The `<icedel:package-confirmations` element can contain one or more `<icedel:confirmation`. See Figure 5.26.

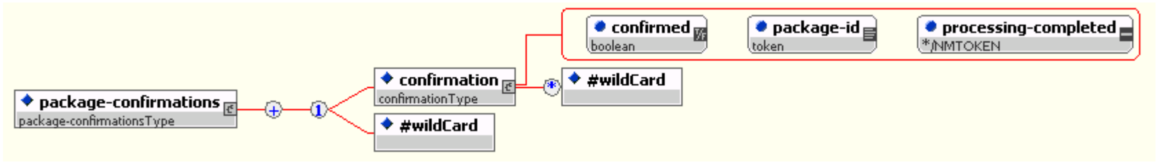


Figure 5.26 Package Confirmations Structure

Each `<icedel:confirmation` has the following attributes:

- **confirmed**
  Required. This attribute specifies whether the package delivery is confirmed. The value is Boolean and there is no default.
- **package-id**
  Required. This attribute specifies the unique id of the package within a subscription for which delivery is confirmed.
- **processing-completed**
  Optional. This attribute specifies whether the package was simply received or whether it was processed. Values are “`received`” and “`processed`”. There is no default.

Chapter 6. Extending the ICE Protocol

6.1 Overview

Authors of the ICE 2.0 Specification have purposely limited its scope to define Basic ICE and Full ICE. Advanced syndication operations are allowed for as extensions to ICE 2.0. This extended level of ICE conformance is known as *Optional ICE*. Optional ICE extensions allow implementers to extend the ICE protocol in such a way that advanced syndication operations may be allowed for in a predictable and controlled manner and interoperation can be achieved.

This chapter describes how to extend the ICE protocol. It provides details about how the use XML Namespaces along with custom WSDL scripts can be used to extend the ICE protocol.

6.2 More About XML 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 while preventing element collisions and enabling computers to unequivocally determine exact points of reference. Such unique addressing is critical to support extensibility of ICE 2.0.

In ICE 2.0, all ICE-defined elements found in one of three ICE namespaces to enable ICE to function as a Web service and utilize SOAP messaging. In addition, an ICE namespace for simple datatypes has been defined.

The ICE 2.0 namespaces include:

- xmlns:icesdt = " http://icestandard.org/ICE/Spec/V20/simpledatatypes"
- 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

6.2.1 Using XML Namespaces in an ICE Message

The following example shows how elements from different namespaces are combined in a simple ICE message. Notice here that we are accessing the W3C SOAP envelope namespace (xmlns:env) as well as elements from the ICE message namespace (xmlns:icemes) and the ICE delivery namespace (xmlns:icedel). Each namespace is shown in bold.

```
<?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 is-new="true">
          <icedel:item-ref item-id="xx203"/>
           <icedel:reference
                 url="http://mysite.com/xxx.htm">
        </icedel:add>
    </icedel:package>
  </env:Body>
<env:Envelope>
```

6.2.2 Defining Extensions

Extensions to ICE 2.0 can be made in a very simple, yet XML-conformant manner or in a more formal manner. These types of ICE extensions are discussed in Section 6.2.2.1 and 6.2.2.2.

6.2.2.1 Simple ICE Extensions

Simple ICE extensions are very straightforward. They do not require the definition of an XML schema or writing new WSDL scripts. For simple ICE extensions just use your own attributes and elements, within a unique namespace, in a well-formed SOAP/ICE instance.

6.2.2.1.1 Select a Namespace

If you are going to mix your own elements and attributes with the ICE message, you must select a namespace to identify the elements. That namespace will be declared the first time you use the element. Note how this is done in the example:

```
<icemes:status-code code="200">
  <myice:codeDescription
     xmlns:myice = "http://myco.com/myice">
```

It is important to understand that the main purpose of a namespace name such as http://myco.com/myice is a URI but not meant to point to a resource. Rather it is used to provide unique identification. The namespace name is not required to be de-referencable. So in the case of a simple ICE extension, this namespace name need not point to anything. And you do not have to have an XML schema definition behind it.

6.2.2.1.2 Use Your Own Elements

While the ICE Authoring Group had the ability to add extensions to ICE 2.0 as a design goal, they also were committed to limiting extensions to those that were created in a predictable and controlled manner. To accomplish this goal, designers of ICE 2.0 created points in the ICE structure where extensions from other namespaces could be added. In Figure 6.1, you can see where the ICE header may be extended. In the positions where `#wildCard` appears, elements from any namespace can be included.

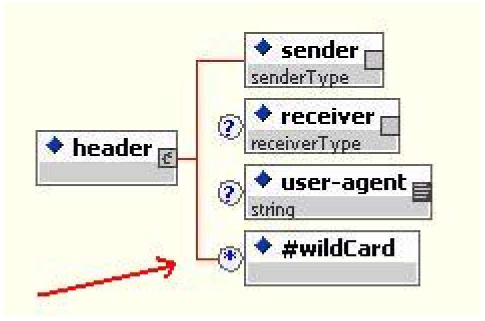


Figure 6.1 ICE extensions allowed at limited points

Once you have declared the namespace, you are free to use elements from that namespace any place that ICE 2.0 allows extensions.

6.2.2.1.3 Use Your Own Attributes

In addition to using your own elements, you may want to extend ICE by adding attributes as well. The Authors of ICE 2.0 have accounted for this in their schemas. Each attribute list within an ICE schema contains a mechanism that enables you to add any attribute from any namespace. It provides a sort of attribute wildcard so that you may add attributes from any namespace. The `xs:anyAttribute` mechanism is highlighted in the example schema below:

```
<xs:element name = "cancellation">
  <xs:complexType>
    <xs:attribute name = "cancellation-id" use = "required"
type = "xs:token"/>
    <xs:attribute name = "subscription-id" use = "required"
type = "xs:token"/>
    <xs:anyAttribute namespace = "##other" processContents
= "lax"/>
  </xs:complexType>
</xs:element>
```

When you add your own attributes, remember to preface the attribute with the namespace, just as you did with the elements.

```
<icemes:status-code xmlns:myice = “http://myco.com/myice”
   code=”200” myice:type=”success”>
   <myice:codeDescription>Description of the code goes here
   </myice:codeDescription>
 </icemes:status-code>
```

Note that the namespace declaration can be used within any element start-tag and the scope of the namespace is within that element. So in the example of using your own elements, the scope is simply within `<myice:codeDescription`. But in this example, the scope of the namespace is anywhere within the `<icemes:status-code`.

6.2.2.2 Formal ICE Extensions

There is a specific method to defining formal extensions to ICE 2.0.

1. Declare an XML schema in a unique namespace for the extensions you wish to implement
2. Use your own elements
3. Use your own attributes
4. Create new WSDL scripts to reflect the extensions, if necessary

6.2.2.2.1 Declare Your Own XML Schema

If you wish to extend ICE 2.0 by adding elements with new functionality, such as to support parameter negotiation, begin by defining your own XML schema. The example below shows a hypothetical schema where we have added the element <`accessCode`.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns = "http://myco.com/myschema/extendheader"
  targetNamespace = "http://myco.com/myschema/extendheader"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    elementFormDefault = "qualified">
  <xs:include schemaLocation = "ice-simpledatatypes.xsd"/>
  <xs:element name = "accessCode">
    <xs:complexType>
      <xs:attribute name = "code" use = "required" type =
        "xs:token"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6.2.2.2.2 Using Your Own Elements

Just as with simple ICE extensions, you are free to use your own elements wherever ICE 2.0 allows extensions. This will be covered in Section 6.3.

NOTE: If you have declared your own schema, it is expected that you not only provide well-formed XML, but that the elements will validate according to your schema.

6.2.2.2.3 Using Your Own Attributes

Just as with simple ICE extensions, you are free to use your own attributes on any element. You must, however, use proper namespace conventions.

6.3 Where Can ICE be Extended?

As was pointed out earlier, the authors of ICE 2.0 were careful to allow for extensions in places that were predictable and controlled. These extension points apply whether ICE is simply extended or formally extended. Graphics of the ICE 2.0 schema show where Full ICE can be extended by either method.

6.3.1 Extensions in the ICE Message

The ICE Message is made up of the ICE header and of ICE status codes. Using namespaces, we are able to include these within the SOAP envelope header and body respectively. The ICE header can be extended by adding elements to the

`<icemes:header` itself. See Figure 6.1. Note that extensions may be made wherever `#wildcard` is shown.

The ICE status code is shown in Figure 6.2. Notice that we can add our own extension elements to the status code.

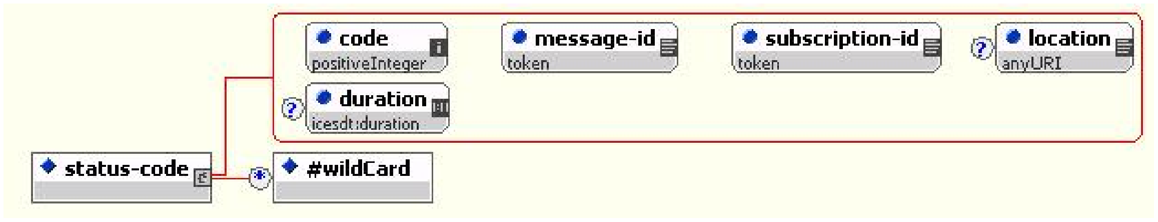


Figure 6.2 ICE status-code extensions

In this example, we have added an element from our own namespace, “`myice:`” to the status code to add a description field:

```
<icemes:status-code code=”202”
       reason=”Package sequence state already current”
       subscription-id=”KKK12U03”
       xmlns:myice = “http://myco.com/myice”>
   <myice:description>This code indicates that the
subscription status is up to date</myice:description>
</icemes:status-code>
```

6.3.2 Extensions in ICE Delivery

Elements within the ICE delivery namespace have also been designed to allow for controlled extensions. The ICE package shown in Figure 6.3 can be extended directly or within a `<icedel:group`, `<icedel:add`, or `<icedel:remove-item`.

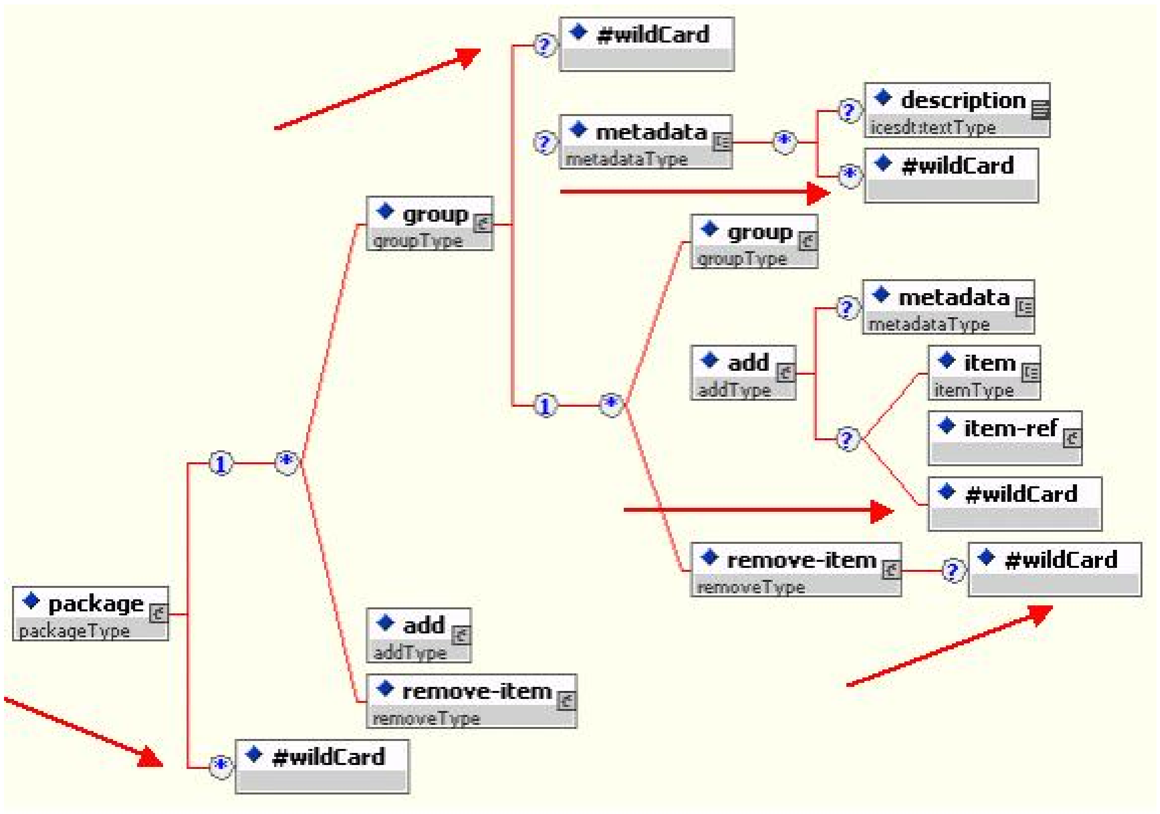


Figure 6.3 Extensions may be made at several locations within the package elements

In the following example an Ice package has been extended to include `<myice:removal-reason` from the namespace “`myice:`”.

```
<icedel:package
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     new-state="ICE-ANY"
     old-state="ICE-ANY"
     fullupdate="true"
     package-id="16T00U9"
     subscription-id="KKK0098">
  <icedel:remove-item subscription-element-id=”K1234D”
     xmlns:myice = “http://myco.com/myice”>
        <myice:removal-reason>This item is out-of-date
           and has no replacement
        </myice:removal-reason>
  </remove-item>
</icedel:package>
```

Note: It turns out this very extension mechanism enables the Syndicator to include `<icesub:offer` within a package in response to the `<icesub:get-catalog` request from a Subscriber. In this case the Syndicator is including the offer from the `icesub:` namespace within the `<icedel:add` element. But extensions to other namespaces can be made at the same nodes as well.

6.3.3 Extensions in ICE Subscribe

Elements within the ICE subscribe namespace have also been designed to allow for controlled extensions. See Figure 6.4 to find extension nodes.

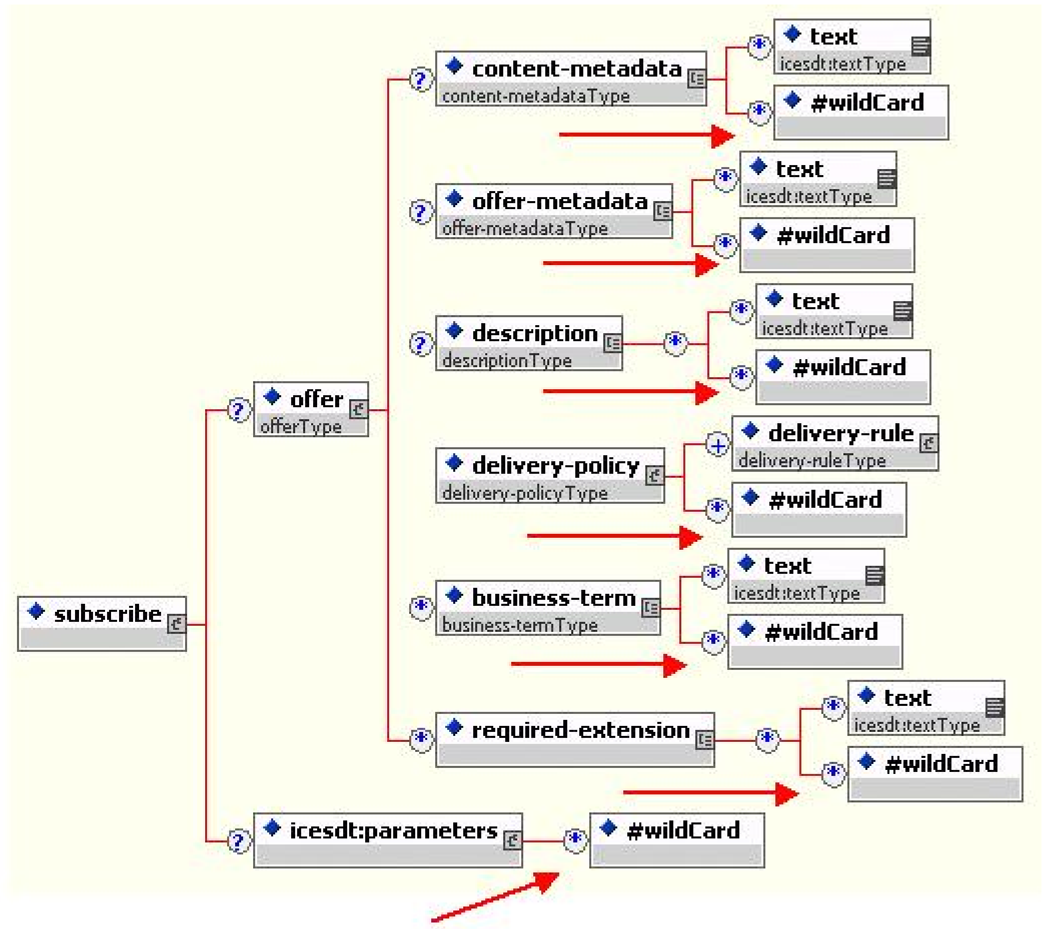


Figure 6.4 Extensions to the ICE subscription mechanism

In the following example note how the subscribe message has been extended. A new namespace "`myice:`" has been declared. Then an element from within that namespace (`<myice:description-code`) has been added to `<icesub:description` in order to enable the use of standard description coding.

```
<icesub:subscribe>
    <icesub:offer
    xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
    offer-id="offID2"
    name="offName2">
      <icesub:description
        xmlns:myice = “http://myco.com/myice”>
          <myice:description-code code=”C1212”/>
      </icesub:description>
      <icesub:delivery-policy quantity="100"
        expiration-priority="quantity">
         <icesub:delivery-rule mode="push">
          <icesub: transport protocol="soap"
              packaging-style=”ice”>
            <icesub:delivery-endpoint
              url="http://sub.com/push.jsp" username="foo"
              password="foofoo"/>
           </icesub:transport>
         </icesub:delivery-rule>
       </icesub:delivery-policy>
    </icesub:offer>
</icesub:subscribe>
```

6.4 Indicating ICE Extensions

When a Syndicator is extending ICE, there are several indications given to the Subscriber. These include:

6.4.1 ICE Message Header

The `<icemes:header` includes attributes for both the `<icemes:sender` and `<icemes:receiver` to indicate their compliance level. This attribute has pre-defined values of “basic” and “full” with a default of “basic”. However the value of the attribute is anyURI, and enables the sender to point to a URI that defines the Optional ICE extensions. See Figure 6.5.

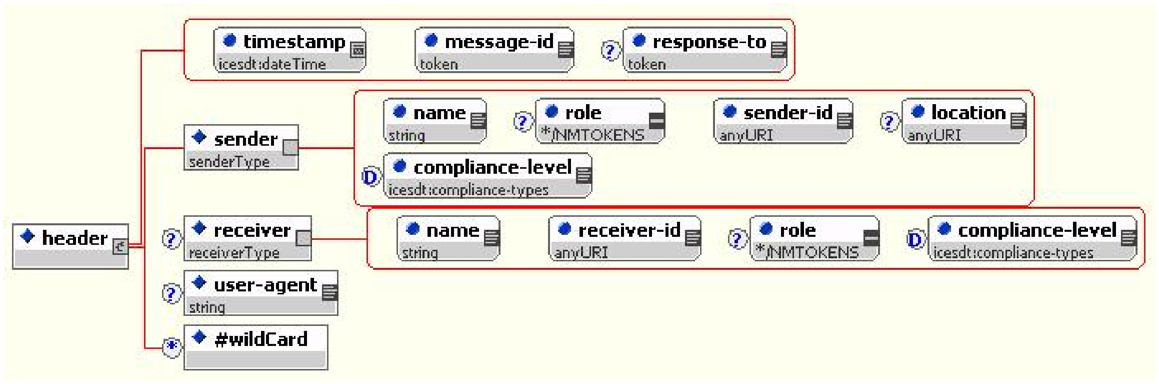


Figure 6.5 Compliance Level Attribute in ICE Message Header

6.4.2 ICE Offer

Extended ICE is indicated in the offer using the <icesub:required-extensions element. See Figure 6.6.

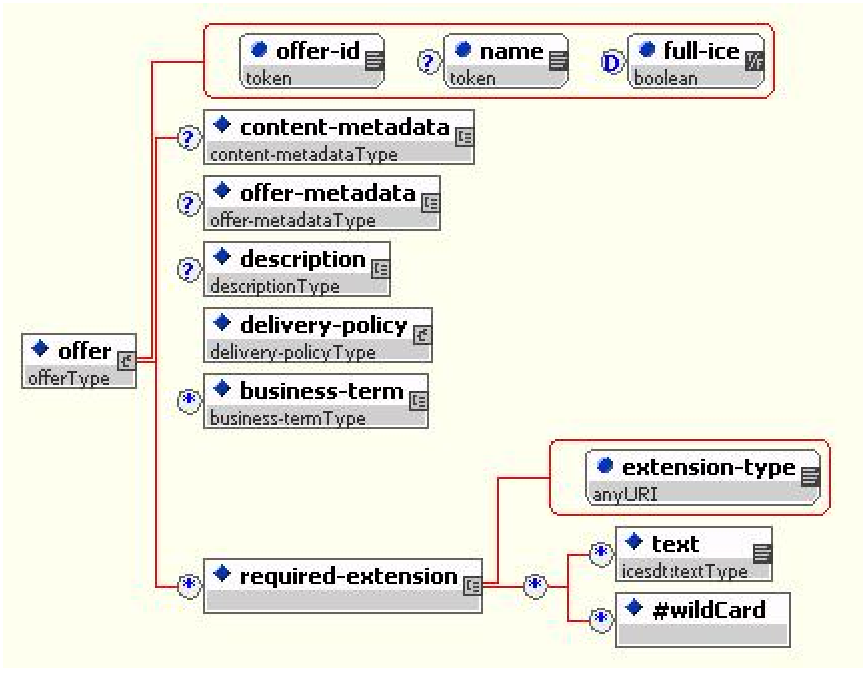


Figure 6.6 Required Extensions indicated in Offer

> Note: Extensions assume Full ICE capabilities. So in addition to specifying one or more required extensions in the offer, the Syndicator must have the `full-ice=` attribute set to "`true`".

6.5 Extending ICE 2.0 to Include ICE 1.* Features

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. A Full ICE implementation implements all the features of the ICE 2.0 specification. The ICE AG chose to remove a number of capabilities of ICE 1.* specification within the ICE 2.0 specification in order to simplify the specification and facilitate implementation. It is the goal of the ICE Authoring Group that features that go beyond Full ICE as defined by this document will be defined by additional specifications.

If an implementer requires features from the ICE 1.* specification, these can be added by the following steps:

1. Examine the structure of the optional feature in the ICE 1.* specification and determine where that optional feature will fit within the extension mechanism for ICE 2.0.
2. Define the optional feature using XML Schema and assign a namespace
3. Reference this optional feature schema along with the ICE schema definitions using XML namespaces
4. Review the WSDL scripts and, if necessary, create two new WSDL scripts to support your Optional ICE implementation:
   - `ice-syndicator-extension.wsdl`
   - `ice-subscriber-extension.wsdl`
5. Implement the Optional Syndicator and Subscriber in ICE software

6.6 Interoperability of ICE Extensions

ICE 2.0 defines three levels of conformance that spell out the features of ICE that must be supported for that level of conformance. These conformance levels are:

- Basic ICE (the default)
- Full ICE (documented in this specification)
- Optional ICE Extensions (Full ICE plus user defined capabilities)

In terms of interoperability

- 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.
- Extended ICE software can be expected to interoperate with other software that understands and supports the same extensions.

> Note: When an ICE implementation with an optional ICE extension is interoperating with another ICE implementation without that extension, it MUST restrict itself to function without that extension.

Appendix A. ICE Simple Datatypes Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://icestandard.org/ICE/V20/simpledatatypes"
       xmlns:icesdt="http://icestandard.org/ICE/V20/simpledatatypes"
       elementFormDefault="qualified"
       attributeFormDefault="unqualified">

  <xs:annotation>
    <xs:documentation>
      We define our own schema for the XML namespace because the
      canonical one available at http://www.w3.org/XML/1998/namespace
doesn't include xml:base and is not in sync with the Recommended
version of W3C Schema 1.0
    </xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="xml.xsd"/>

   <!--
     |
     | attlist.genericmetadata
     |
   -->
  <xs:attributeGroup name="attlist.genericmetadata">
    <xs:annotation>
      <xs:documentation>
        changes from 1.1: moved content-transfer-encoding to item only
      </xs:documentation>
    </xs:annotation>
    <xs:attribute name="atomic-use"
    type="xs:boolean" default="false"/>
    <xs:attribute name="editable"      type="xs:boolean"
default="false"/>
    <xs:attribute name="ip-status" type="xs:token"/>
    <xs:attribute name="license"type="xs:token"/>
    <xs:attribute name="rights-holder" type="xs:token"/>
    <xs:attribute name="show-credit"   type="xs:boolean"/>
    <xs:attribute name="item-type" type="icesdt:item-types"
       default="http://icestandard.org/ICE/V20/item-type/undefined"/>
  </xs:attributeGroup>

   <!--
     | ice-compliance
     | Basic ICE compliance:
         http://icestandard.org/ICE/V20/syndicator/basic
     | Full  ICE compliance:
         http://icestandard.org/ICE/V20/syndicator/full
     |
   -->
  <xs:simpleType name="compliance-types">
```

```
    <xs:annotation>
      <xs:documentation>
        a URI that names an ICE compliance level.
      </xs:documentation>
    </xs:annotation>
        <xs:union memberTypes="xs:anyURI">
      <xs:simpleType>
        <xs:restriction base="xs:anyURI">
          <xs:enumeration
            value="http://icestandard.org/ICE/V20/syndicator/basic"/>
          <xs:enumeration
            value="http://icestandard.org/ICE/V20/syndicator/full"/>
          <xs:enumeration
            value="http://icestandard.org/ICE/V20/itemtype/undefined"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:anyURI"/>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>

<!--
     |
     | item-types (feature)
     |
     |
   -->
  <xs:simpleType name="item-types">
    <xs:annotation>
      <xs:documentation>
        a URI that defines the content of an item.
      </xs:documentation>
    </xs:annotation>
        <xs:union memberTypes="xs:anyURI">
      <xs:simpleType>
        <xs:restriction base="xs:anyURI">
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/undefined"/>
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/offer"/>
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/rss0.91"/>
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/rss0.92"/>
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/rss1.0"/>
          <xs:enumeration
           value="http://icestandard.org/ICE/V20/item-type/rss2.0"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:anyURI"/>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
```

```
 <!--
   |
   | dateTime
   |
 -->
<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="xs:dateTime">
    <xs:pattern value=".*Z"/>
  </xs:restriction>
</xs:simpleType>



 <!--
   |
   | time
   |
 -->
<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="xs:time">
    <xs:pattern value=".*Z"/>
  </xs:restriction>
</xs:simpleType>



 <!--
   |
   | duration
   |
 -->
<xs:simpleType name="duration">
  <xs:annotation>
    <xs:documentation>
      the pattern here expresses the restriction that
      durations in ICE must only include seconds
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:duration">
    <xs:pattern value="PT\d*\.?\d*S"/>
  </xs:restriction>
</xs:simpleType>
```

```
 <!--
   |
   | parameters
   |
 -->
<xs:element name="parameters">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace="##other" processContents="lax"
              minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
</xs:element>



<!--
  |
  | package-sequence-stateType
  |
  -->
<xs:simpleType name="package-sequence-stateType">
  <xs:union memberTypes="xs:token">
    <xs:simpleType>
      <xs:restriction base="xs:token">
        <xs:enumeration value="ICE-INITIAL"/>
        <xs:enumeration value="ICE-ANY"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:token"/>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>



<!--
  |
  | textType
  |
 -->
<xs:complexType name="textType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
   <!--
     |
     | urlAccessType
     |
   -->
  <xs:complexType name="urlAccessType">
    <xs:annotation>
      <xs:documentation>
        changes from 1.1:
          * removed the (FIXED) ice-element attribute
          * added link-only attribute
          * remove access-window
      </xs:documentation>
    </xs:annotation>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:any namespace="##other" processContents="lax"
              minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="url"  use="required"   type="xs:anyURI"/>
    <xs:attribute name="username"              type="xs:token"/>
    <xs:attribute name="password"              type="xs:token"/>
    <xs:attribute name="authentication-scheme">
      <xs:simpleType>
        <xs:union memberTypes="xs:string">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="basic"/>
              <xs:enumeration value="digest"/>
            </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
            <xs:restriction base="xs:string"/>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>

</xs:schema>
```

Appendix B. ICE Message Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "http://icestandard.org/ICE/V20/message"
   xmlns:icemsg = "http://icestandard.org/ICE/V20/message"
   xmlns:xs = "http://www.w3.org/2001/XMLSchema"
   xmlns:icesdt = "http://icestandard.org/ICE/V20/simpledatatypes"
   elementFormDefault = "qualified">

  <import namespace =
        "http://icestandard.org/ICE/V20/simpledatatypes"
        schemaLocation = "ice-simpledatatypes.xsd"/>


  <!--
    |
    | header
    |
   -->
  <element name = "header">
    <complexType>
      <sequence>
        <element name = "sender" type = "icemsg:senderType"/>
        <element name = "receiver"
           type = "icemsg:receiverType" minOccurs = "0"/>
        <element ref = "icemsg:user-agent" minOccurs = "0"/>
        <any namespace = "##other" processContents = "lax"
           minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "timestamp" use = "required"
          type = "icesdt:dateTime"/>
      <attribute name = "message-id" use = "required"
          type = "token"/>
      <attribute name = "response-to"
          type = "token"/>
      <anyAttribute namespace = "##other" processContents = "lax"/>
    </complexType>
  </element>

  <!--
    |
    | receiverType
    |
   -->

  <complexType name = "receiverType">
    <attribute name = "name" use = "required" type = "string"/>
    <attribute name = "receiver-id" use = "required" type = "anyURI"/>
    <attribute name = "role">
      <simpleType>
        <restriction base = "NMTOKENS">
          <enumeration value = "subscriber"/>
          <enumeration value = "syndicator"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name = "compliance-level" default = "basic"
        type = "icesdt:compliance-types"/>
    <anyAttribute namespace = "##other" processContents = "lax"/>
  </complexType>

  <!--
    |
    | senderType
    |
   -->

  <complexType name = "senderType">
    <attribute name = "name" use = "required" type = "string"/>
    <attribute name = "role">
      <simpleType>
        <restriction base = "NMTOKENS">
          <enumeration value = "subscriber"/>
          <enumeration value = "syndicator"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name = "sender-id" use = "required" type = "anyURI"/>
    <attribute name = "location" type = "anyURI"/>
    <attribute name = "compliance-level" default = "basic" type =
"icesdt:compliance-types"/>
    <anyAttribute namespace = "##other" processContents = "lax"/>
  </complexType>

  <!--
    |
    | user-agent
    |
   -->

      <element name = "user-agent" type = "icesdt:textType"/>

    <!--
    | ping request message
    -->

  <element name = "ping">
    <complexType/>
  </element>

  <!–
   |
   | OK simple confirmation response message
   |
  -->

  <element name = "OK">
    <complexType/>
  </element>
  <!—
   |
   |   Status-code
   |
  -->

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

Appendix C. ICE Delivery Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "http://icestandard.org/ICE/V20/delivery"
   xmlns:icedel = "http://icestandard.org/ICE/V20/delivery"
   xmlns:xs = "http://www.w3.org/2001/XMLSchema"
   xmlns:icesdt = "http://icestandard.org/ICE/V20/simpledatatypes"
   elementFormDefault = "qualified">
  <import namespace = “http://icestandard.org/ICE/V20/simpledatatypes”
      schemaLocation = "ice-simpledatatypes.xsd"/>
  <import namespace = "http://www.w3.org/XML/1998/namespace"
schemaLocation = "xml.xsd"/>
  <attributeGroup name = "attlist.item">
    <attribute name = "content-filename" type = "token"/>
    <attribute name = "content-type"
        default = "application/octet-stream" type = "token"/>
  </attributeGroup>
  <group name = "cm.package">
    <choice>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element name = "group" type = "icedel:groupType"/>
        <element name = "add" type = "icedel:addType"/>
        <element ref = "icedel:remove-item"/>
      </choice>
    </choice>
  </group>

  <!--
    |
    | get-packages
    |
   -->

  <element name = "get-packages">
    <complexType>
      <sequence>
        <element ref = "icedel:get-package" maxOccurs = "unbounded"/>
        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
      </sequence>
      <anyAttribute namespace = "##other" processContents = "lax"/>
    </complexType>
  </element>

  <!--
    |
    | get-package
    |
   -->

  <element name = "get-package" type = "icedel:get-packageType"/>

  <!--
```

```
     |
     | get-packageType
     |
   -->

   <complexType name = "get-packageType">
     <sequence>
       <element ref = "icesdt:parameters" minOccurs = "0"/>
     </sequence>
     <attribute name = "current-state" type = "token"/>
     <attribute name = "subscription-id" use = "required" type =
 "token"/>
     <anyAttribute namespace = "##other" processContents = "lax"/>
   </complexType>

   <!--
     |
     | packages
     |
    -->

   <element name = "packages">
     <complexType>
       <sequence minOccurs = "0" maxOccurs = "unbounded">
         <element ref = "icedel:package"/>
         <any namespace = "##other" processContents = "lax" minOccurs =
 "0" maxOccurs = "unbounded"/>
       </sequence>
       <anyAttribute namespace = "##other" processContents = "lax"/>
     </complexType>
   </element>

   <!--
     |
     | package
     |
    -->

   <element name = "package" type = "icedel:packageType"/>

   <!--
     |
     | packageType
     |
    -->

   <complexType name = "packageType">
     <sequence>
       <group ref = "icedel:cm.package"/>
       <any namespace = "##other" processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
     </sequence>
     <attribute name = "package-id" use = "required" type = "token"/>
     <attribute name = "subscription-id" use = "required"
          type = "token"/>
     <attribute name = "fullupdate" default = "true"
          type = "boolean"/>
```

```
  <attribute name = "confirmation" default = "false"
     type = "boolean"/>
  <attribute name = "new-state" default = "ICE-ANY"
     type = "icesdt:package-sequence-stateType"/>
  <attribute name = "old-state" default = "ICE-ANY"
      type = "icesdt:package-sequence-stateType"/>
  <anyAttribute namespace = "##other" processContents = "lax"/>
</complexType>

<!--
  |
  | groupType
  |
 -->

<complexType name = "groupType">
  <sequence>
    <any namespace = "##other" processContents = "lax"
       minOccurs = "0"/>
    <element name = "metadata" type = "icedel:metadataType"
       minOccurs = "0"/>
    <group ref = "icedel:cm.package"/>
  </sequence>
  <attribute name = "name" type = "token"/>
  <attribute name = "subscription-element-id" type = "string"/>
</complexType>

<!--
  |
  | addType
  |
  |
 -->

<complexType name = "addType">
  <sequence>
    <element name = "metadata" type = "icedel:metadataType"
        minOccurs = "0"/>
    <choice minOccurs = "0">
      <element name = "item" type = "icedel:itemType"/>
      <element name = "item-ref">
        <complexType>
          <sequence>
            <element name = "reference"
               type = "icesdt:urlAccessType"/>
            <any namespace = "##other"
              processContents = "lax" minOccurs = "0"
              maxOccurs = "unbounded"/>
          </sequence>
          <attribute name = "retrieve-after"
             type = "icesdt:dateTime"/>
          <attribute name = "name" type = "token"/>
        </complexType>
      </element>
      <any namespace = "##other" processContents = "lax"/>
    </choice>
  </sequence>
  <attribute name = "subscription-element-id" type = "token"/>
  <attribute name = "is-new" type = "boolean"/>
  <attribute name = "activation" type = "icesdt:dateTime"/>
  <attribute name = "expiration" type = "icesdt:dateTime"/>
  <anyAttribute namespace = "##other" processContents = "lax"/>
</complexType>

<!--
  |
  | metadataType
  |
 -->

<complexType name = "metadataType" mixed = "true">
  <sequence minOccurs = "0" maxOccurs = "unbounded">
    <element name = "description" type = "icesdt:textType"
         minOccurs = "0"/>
    <any namespace = "##local ##other " processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
  </sequence>
  <attributeGroup ref = "icedel:attlist.item"/>
  <attributeGroup ref = "icesdt:attlist.genericmetadata"/>
  <anyAttribute namespace = "##other" processContents = "lax"/>
</complexType>

<!--
  |
  | itemType
  |
 -->

<complexType name = "itemType" mixed = "true">
  <sequence>
    <any namespace = "##other" processContents = "lax"
        minOccurs = "0" maxOccurs = "unbounded"/>
  </sequence>
  <attribute name = "content-transfer-encoding"
      default = "x-native-xml">
    <simpleType>
      <restriction base = "NMTOKEN">
        <enumeration value = "base64"/>
        <enumeration value = "x-native-xml"/>
      </restriction>
    </simpleType>
  </attribute>
  <attribute name = "name" type = "token"/>
</complexType>


<!--
  |
  | remove-item
  |
 -->

<element name = "remove-item" type = "icedel:removeType"/>
```

```
<!--
  |
  | removeType
  |
 -->

<complexType name = "removeType">
  <sequence>
    <any namespace = "##other" processContents = "lax"
      minOccurs = "0"/>
  </sequence>
  <attribute name = "subscription-element-id" use = "required"
     type = "token"/>
  <anyAttribute namespace = "##other" processContents = "strict"/>
</complexType>

<!--
  |
  | package-confirmations
  |
 -->

<element name = "package-confirmations"
      type = "icedel:package-confirmationsType"/>

<!--
  |
  | package-confirmationsType
  |
 -->

<complexType name = "package-confirmationsType">
  <sequence maxOccurs = "unbounded">
    <choice>
      <element name = "confirmation"
         type = "icedel:confirmationType"/>
      <any namespace = "##other" processContents = "lax"/>
    </choice>
  </sequence>
</complexType>

<!--
  |
  | confirmationType
  |
 -->

<complexType name = "confirmationType">
  <sequence>
    <any namespace = "##other" processContents = "lax"
        minOccurs = "0" maxOccurs = "unbounded"/>
  </sequence>
  <attribute name = "confirmed" use = "required" type = "boolean"/>
  <attribute name = "package-id" use = "required" type = "token"/>
  <attribute name = "processing-completed" use = "required">
    <simpleType>
      <restriction base = "NMTOKEN">
```

```
          <enumeration value = "received"/>
          <enumeration value = "processed"/>
        </restriction>
      </simpleType>
    </attribute>
    <anyAttribute namespace = "##other" processContents = "lax"/>
  </complexType>
</schema>
```

Appendix D. ICE Subscribe Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns = "http://icestandard.org/ICE/V20/subscribe"
   targetNamespace = "http://icestandard.org/ICE/V20/subscribe"
   xmlns:xs = "http://www.w3.org/2001/XMLSchema"
   xmlns:icesdt = "http://icestandard.org/ICE/V20/simpledatatypes"
   elementFormDefault = "qualified">
  <xs:import namespace =
      "http://icestandard.org/ICE/V20/simpledatatypes"
       schemaLocation = "ice-simpledatatypes.xsd"/>

  <!--
    |
    | offer
    |
   -->

  <xs:element name = "offer" type = "offerType"/>

  <!--
    |
    | offerType
    |
   -->

  <xs:complexType name = "offerType">
    <xs:sequence>
      <xs:element name = "content-metadata"
         type = "content-metadataType" minOccurs = "0"/>
      <xs:element name = "offer-metadata"
          type = "offer-metadataType" minOccurs = "0"/>
      <xs:element name = "description"
          type = "descriptionType" minOccurs = "0"/>
      <xs:element name = "delivery-policy"
          type = "delivery-policyType"/>
      <xs:element name = "business-term"
          type = "business-termType" minOccurs = "0"
          maxOccurs = "unbounded"/>
      <xs:element name = "required-extension" minOccurs = "0"
          maxOccurs = "unbounded">
        <xs:complexType>
          <xs:complexContent>
            <xs:extension base = "required-extensionType">
              <xs:attribute name = "extension-type"
                 use = "required" type = "xs:anyURI"/>
            </xs:extension>
          </xs:complexContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name = "offer-id" use = "required"
       type = "xs:token"/>
    <xs:attribute name = "name" type = "xs:token"/>
    <xs:attribute name = "valid-after" type = "xs:dateTime"/>
    <xs:attribute name = "expiration-date" type = "xs:dateTime"/>
    <xs:attribute name = "full-ice" default = "false"
       type = "xs:boolean"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>

  <!--
    |
    | delivery-policyType
    |
   -->

  <xs:complexType name = "delivery-policyType">
    <xs:sequence>
      <xs:element name = "delivery-rule" type = "delivery-ruleType"
         maxOccurs = "unbounded"/>
      <xs:any namespace = "##other" processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "startdate" type = "icesdt:dateTime"/>
    <xs:attribute name = "stopdate" type = "icesdt:dateTime"/>
    <xs:attribute name = "quantity" type = "xs:integer"/>
    <xs:attribute name = "expiration-priority" default = "first">
      <xs:simpleType>
        <xs:restriction base = "xs:NMTOKEN">
          <xs:enumeration value = "first"/>
          <xs:enumeration value = "time"/>
          <xs:enumeration value = "quantity"/>
          <xs:enumeration value = "last"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>

  <!--
    |
    | delivery-ruleType
    |
   -->

  <xs:complexType name = "delivery-ruleType">
    <xs:sequence>
      <xs:element name = "transport" maxOccurs = "unbounded"
         minOccurs = "1" type = "transportType"/>
      <xs:any namespace = "##local ##other"
          processContents = "lax" minOccurs = "0"
          maxOccurs = "unbounded"/>
    </xs:sequence>

    <xs:attribute name = "mode" default = "pull">
      <xs:simpleType>
        <xs:restriction base = "xs:NMTOKEN">
```

```
          <xs:enumeration value = "pull"/>
          <xs:enumeration value = "push"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name = "monthday" type = "xs:NMTOKENS"/>
    <xs:attribute name = "weekday" type = "xs:NMTOKENS"/>
    <xs:attribute name = "startdate" type = "icesdt:dateTime"/>
    <xs:attribute name = "stopdate" type = "icesdt:dateTime"/>
    <xs:attribute name = "starttime" type = "icesdt:time"/>
    <xs:attribute name = "duration" type = "icesdt:duration"/>
    <xs:attribute name = "min-num-updates" type = "xs:integer"/>
    <xs:attribute name = "max-num-updates"
         type = "xs:integer"/>
    <xs:attribute name = "incremental-update" type = "xs:boolean"
         default = "false"/>
    <xs:attribute name = "required" type = "xs:boolean"
         default = "true"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>

<!--
    |
    | transportType
    |
   -->

  <xs:complexType name = "transportType" mixed = "true">
      <xs:sequence>
        <xs:element name = "delivery-endpoint"
            type = "icesdt:urlAccessType" minOccurs = "0"
            maxOccurs = "1" />
    </xs:sequence>
     <xs:attribute name = "protocol" default = "http:get">
      <xs:simpleType>
              <xs:restriction base = "xs:NMTOKEN">
                <xs:enumeration value = "http:get"/>
                <xs:enumeration value = "ftp"/>
                <xs:enumeration value = "mailto"/>
                <xs:enumeration value = "soap"/>
              </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name = "packaging-style" default = "ice">
            <xs:simpleType>
              <xs:restriction base = "xs:NMTOKEN">
                <xs:enumeration value = "ice"/>
                <xs:enumeration value = "raw"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
  </xs:complexType>
```

```
<!--
  |
  | content-metadataType
  |
 -->

<xs:complexType name = "content-metadataType" mixed = "true">
  <xs:sequence>
    <xs:element name = "text" type = "icesdt:textType"
       minOccurs = "0" maxOccurs = "unbounded"/>
    <xs:any namespace = "##local ##other" processContents = "lax"
             minOccurs = "0" maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:attributeGroup ref = "icesdt:attlist.genericmetadata"/>
</xs:complexType>

<!--
  |
  | offer-metadataType
  |
 -->

<xs:complexType name = "offer-metadataType" mixed = "true">
  <xs:sequence>
    <xs:element name = "text" type = "icesdt:textType"
         minOccurs = "0" maxOccurs = "unbounded"/>
    <xs:any namespace = "##local ##other " processContents = "lax"
        minOccurs = "0" maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace = "##other" processContents = "lax"/>
</xs:complexType>

<!--
  |
  | descriptionType
  |
 -->

<xs:complexType name = "descriptionType" mixed = "true">
  <xs:sequence minOccurs = "0" maxOccurs = "unbounded">
    <xs:element name = "text" type = "icesdt:textType"
       minOccurs = "0" maxOccurs = "unbounded"/>
    <xs:any namespace = "##local ##other " processContents = "lax"
       minOccurs = "0" maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace = "##other" processContents = "lax"/>
</xs:complexType>

<!--
  |
  | business-termType
  |
 -->

<xs:complexType name = "business-termType" mixed = "true">
  <xs:sequence>
    <xs:element name = "text" type = "icesdt:textType"
```

```
        minOccurs = "0" maxOccurs = "unbounded"/>
      <xs:any namespace = "##other" processContents = "lax"
        minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "type" use = "required">
      <xs:simpleType>
        <xs:restriction base = "xs:NMTOKEN">
          <xs:enumeration value = "credit"/>
          <xs:enumeration value = "licensing"/>
          <xs:enumeration value = "payment"/>
          <xs:enumeration value = "reporting"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name = "url" type = "xs:anyURI"/>
    <xs:attribute name = "name" type = "xs:token"/>
    <xs:attribute name = "usage-required" type = "xs:boolean"/>
    <xs:attribute name = "business-term-id" type = "xs:string"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>

  <!--
    |
    | required-extensionType
    |
   -->

  <xs:complexType name = "required-extensionType" mixed = "true">
    <xs:sequence minOccurs = "0" maxOccurs = "unbounded">
      <xs:element name = "text" type = "icesdt:textType"
         minOccurs = "0" maxOccurs = "unbounded"/>
      <xs:any namespace = "##local ##other " processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>

  <!--
    |
    | offer-metadataType
    |
   -->

  <xs:element name = "subscribe">
    <xs:complexType>
      <xs:sequence>
        <xs:element name = "offer" type = "offerType" minOccurs = "0"/>
        <xs:element ref = "icesdt:parameters" minOccurs = "0"/>
      </xs:sequence>
      <xs:attribute name = "subscription-name" type = "xs:token"/>
      <xs:attribute name = "offer-id" type = "xs:token"/>
      <xs:anyAttribute namespace = "##other" processContents = "lax"/>
    </xs:complexType>
  </xs:element>
```

```
<!--
  |
  | subscription
  |
 -->

<xs:element name = "subscription" type = "subscriptionType"/>

<!--
  |
  | subscriptionType
  |
 -->

<xs:complexType name = "subscriptionType">
  <xs:sequence>
    <xs:element name = "offer" type = "offerType"/>
    <xs:any namespace = "##other" processContents = "lax"
      minOccurs = "0" maxOccurs = "unbounded"/>
  </xs:sequence>
  <xs:attribute name = "subscription-id" use = "required"
      type = "xs:token"/>
  <xs:attribute name = "subscription-name" type = "xs:token"/>
  <xs:attribute name = "current-state"
     type = "icesdt:package-sequence-stateType"/>
  <xs:attribute name = "quantity-remaining" type = "xs:integer"/>
  <xs:anyAttribute namespace = "##other" processContents = "lax"/>
</xs:complexType>

<!--
  | subscribe-fault
  | response message returned in the soap fault details
  | if a subscribe/change-subscription request fails
 -->

<xs:element name = "subscription-fault">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "offer" minOccurs = "1"/>
      <xs:any namespace = "##other" processContents = "lax"
          minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "code" use = "required"
          type = "xs:positiveInteger"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>
</xs:element>

<!--
  |
  | cancel
  |
 -->

<xs:element name = "cancel">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element name = "reason" type = "icesdt:textType"
         minOccurs = "0" maxOccurs = "unbounded"/>
      <xs:any namespace = "##other" processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "subscription-id" use = "required"
        type = "xs:token"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>
</xs:element>

<!--
  |
  | cancellation
  |
 -->

<xs:element name = "cancellation">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace = "##other" processContents = "lax"
         minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "cancellation-id" use = "required"
         type = "xs:token"/>
    <xs:attribute name = "subscription-id" use = "required"
         type = "xs:token"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>
</xs:element>

<!--
  |
  | get-status
  |
 -->

<xs:element name = "get-status">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace = "##other" processContents = "lax"
          minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
    <xs:attribute name = "subscription-id" type = "xs:token"/>
    <xs:anyAttribute namespace = "##other" processContents = "lax"/>
  </xs:complexType>
</xs:element>

<!--
  |
  | status
  |
 -->

<xs:element name = "status" type = "statusType"/>

<!--
```

```
    |
    | statusType
    |
   -->

  <xs:complexType name = "statusType">
    <xs:sequence>
      <xs:element name = "subscription" type = "subscriptionType"
          minOccurs = "0" maxOccurs = "unbounded"/>
      <xs:any namespace = "##other" processContents = "lax"
          minOccurs = "0" maxOccurs = "unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Appendix E. Full ICE Syndicator WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ice-syndicator-full"
     targetNamespace="http://icestandard.org/ICE/V20/syndicator/full"
     xmlns:tns="http://icestandard.org/ICE/V20/syndicator/full"
     xmlns:icemsg="http://icestandard.org/ICE/V20/message"
     xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!--
  | Import schema definitions
   -->
  <import namespace="http://icestandard.org/ICE/V20/message"
          location="./ice-message.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/subscribe"
          location="./ice-subscribe.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/delivery"
          location="./ice-delivery.xsd"/>


  <!-- ping messages -->
  <message name="ping">
    <part name="pingReq" element="icemsg:ping"/>
  </message>
  <message name="ok">
    <part name="okResp" element="icemsg:OK"/>
  </message>

  <!-- subscribe messages -->
  <message name="subscribe">
    <part name="subscribeReq" element="icesub:offer"/>
  </message>
  <message name="subscription">
    <part name="subscriptionResp" element="icesub:subscription"/>
  </message>

  <!-- cancel messages -->
  <message name="cancel-subscription">
    <part name="cancelReq" element="icesub:cancel"/>
  </message>
  <message name="cancellation">
    <part name="cancellationResp" element="icesub:cancellation"/>
  </message>

  <!-- get-status messages -->
  <message name="get-status">
    <part name="getStatusReq" element="icesub:get-status"/>
  </message>
  <message name="status">
    <part name="statusResp" element="icesub:status"/>
  </message>

  <!-- get-packages messages -->
  <message name="get-packages">
    <part name="getPackagesReq" element="icedel:get-packages"/>
  </message>
  <message name="packages">
    <part name="packagesResp" element="icedel:packages"/>
  </message>

  <!-- get-package messages -->
  <message name="get-package">
    <part name="getPackageReq" element="icedel:get-package"/>
  </message>
  <message name="package">
    <part name="packageResp" element="icedel:package"/>
  </message>

  <!-- confirm messages -->
  <message name="confirm">
    <part name="confirmReq" element="icedel:package-confirmations"/>
  </message>

  <message name="status-code">
    <part name="status-code" element="icemsg:status-code"/>
  </message>

  <message name="subscription-fault">
    <part name="subscription-fault" element="icesub:subscription-
fault"/>
  </message>

  <message name="header">
    <part name="header" element="icemsg:header"/>
  </message>


  <!-- portType definition -->
  <portType name="ice-syndicator-full-portType">

   <operation name="ping">
      <input  message="tns:ping"  name="ping"/>
      <output message="tns:ok"    name="ok"/>
      <fault  message="tns:status-code" name="status-code"/>
    </operation>

   <operation name="subscribe">
      <input  message="tns:subscribe"       name="subscribe"/>
      <output message="tns:subscription"    name="subscription"/>
      <fault  message="tns:subscription-fault" name="subscription-
fault"/>
    </operation>
   <operation name="change-subscription">
      <input  message="tns:subscription"    name="subscription"/>
      <output message="tns:subscription"    name="subscription"/>
      <fault  message="tns:subscription-fault" name="subscription-
fault"/>
    </operation>

    <operation name="cancel-subscription">
      <input  message="tns:cancel"       name="cancel"/>
      <output message="tns:cancellation" name="cancellation"/>
      <fault  message="tns:status-code"        name="status-code"/>
    </operation>

    <operation name="get-status">
      <input  message="tns:get-status" name="get-status"/>
      <output message="tns:status"     name="status"/>
      <fault  message="tns:status-code"      name="status-code"/>
    </operation>

    <!-- delivery operations -->
    <operation name="get-packages">
      <input  message="tns:get-packages" name="get-packages"/>
      <output message="tns:packages"     name="packages"/>
      <fault  message="tns:status-code "        name="status-code"/>
    </operation>

    <operation name="get-package">
      <input  message="tns:get-package" name="get-package"/>
      <output message="tns:package"     name="package"/>
      <fault  message="tns:status-code "       name="status-code"/>
    </operation>

    <operation name="confirm">
      <input  message="tns:confirm" name="confirm"/>
      <output message="tns:ok"      name="ok"/>
      <fault  message="tns:status-code "   name="status-code"/>
    </operation>
  </portType>


  <!--
       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"/>

    <!--
         ping
    -->
    <operation name="ping">
      <soap:operation/>
      <input name="ping">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="ok">
        <soap:body use="literal"
         namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
         namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="fault" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         subscribe
    -->
    <operation name="subscribe">
      <soap:operation/>
      <input name="subscribe">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="subscription">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="subscription-fault">
        <soap:fault name="subscription-fault" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         change-subscription
    -->
    <operation name="change-subscription">
      <soap:operation/>
      <input name="subscription">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="subscription">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
        namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="subscription-fault">
        <soap:fault name="subscription-fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         cancel
    -->
    <operation name="cancel-subscription">
      <soap:operation/>
      <input name="cancel">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="cancellation">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         get-status
    -->
    <operation name="get-status">
      <soap:operation/>
      <input name="get-status">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="status">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
         namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         get-packages
    -->
    <operation name="get-packages">
      <soap:operation/>
      <input name="get-packages">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="packages">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         get-package
    -->
    <operation name="get-package">
      <soap:operation/>
      <input name="get-package">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="package">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>

    <!--
         confirm
    -->
    <operation name="confirm">
```

```
      <soap:operation/>
      <input name="confirm">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </input>
      <output name="ok">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        <soap:headerfault message="status-code" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
          namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </fault>
    </operation>
  </binding>


  <!-- Sample Service -->
  <service name="your-ice-syndicator-full">
    <port name="ice-syndicator-full-portType" binding="tns:ice-
syndicator-full-binding">
      <soap:address location="http://your-ice-server.com/soap-ice"/>
    </port>
  </service>
</definitions>
```

Appendix F. Full ICE Subscriber WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ice-subscriber"
     targetNamespace="http://icestandard.org/ICE/V20/wsdl/subscriber"
     xmlns:tns="http://icestandard.org/ICE/V20/wsdl/subscriber"
     xmlns:icemsg="http://icestandard.org/ICE/V20/message"
     xmlns:icedel="http://icestandard.org/ICE/V20/delivery"
     xmlns:icesub="http://icestandard.org/ICE/V20/subscribe"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- import schema definition -->
  <import namespace="http://icestandard.org/ICE/V20/message"
          location="./ice-message.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/delivery"
          location="./ice-delivery.xsd"/>

  <import namespace="http://icestandard.org/ICE/V20/subscribe"
          location="./ice-subscribe.xsd"/>

  <!-- package messages -->
  <message name="package">
    <part name="packageReq"  element="icedel:package"/>
  </message>
  <message name="package-confirmations">
    <part name="packageResp" element="icedel:package-confirmations"/>
  </message>

  <!-- cancel messages -->
  <message name="cancel-subscription">
    <part name="cancelReq" element="icesub:cancel"/>
  </message>
  <message name="cancellation">
    <part name="cancelResp" element="icesub:cancellation"/>
  </message>

  <message name="status-code">
    <part name="status-code" element="icemsg:status-code"/>
  </message>
  <message name="header">
    <part name="header" element="icemsg:header"/>
  </message>


  <!-- portType definition -->
  <portType name="ice-subscriber-portType">
    <operation name="package">
      <input  message="tns:package"name="package"/>
      <output message="tns:package-confirmations" name="package-
confirmations"/>
      <fault  message="tns:status-code" name="status-code"/>
    </operation>

    <operation name="cancel-subscription">
      <input  message="tns:cancel"       name="cancel"/>
      <output message="tns:cancellation" name="cancellation"/>
      <fault  message="tns:status-code" name="status-code"/>
    </operation>
  </portType>


  <!--
       SOAP Binding
  -->
  <binding name="ice-subscriber-binding" type="tns:ice-subscriber-
portType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <!--
         package
    -->
    <operation name="package">
      <soap:operation/>
      <input name="package">
        <soap:body use="literal"
namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </input>
      <output name="package-confirmations">
        <soap:body use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" parts="header" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
        </output>
        <fault name="status-code">
           <soap:fault name="status-code " use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        </fault>
    </operation>

    <!--
         cancel
    -->
    <operation name="cancel-subscription">
      <soap:operation/>
      <input name="cancel">
        <soap:body use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </input>
      <output name="cancellation">
```

```
        <soap:body use="literal"
         namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:header message="header" parts="header" use="literal"
           namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
        <soap:headerfault message="fault" use="literal"
           namespace="http://icestandard.org/ICE/V20/syndicator/full"/>
      </output>
      <fault name="status-code">
        <soap:fault name="status-code" use="literal"
          namespace="http://icestandard.org/ICE/V20/wsdl/subscriber"/>
      </fault>
    </operation>
  </binding>


  <!-- Sample Service -->
  <service name="your-ice-subscriber">
    <port name="ice-subscriber-portType" binding="tns:ice-subscriber-
binding">
     <soap:address location="http://your-ice-server.com/soap-ice"/>
    </port>
  </service>
</definitions>
```

Appendix G. ICE Package Sequence Model

Incremental package delivery in Full ICE follows a *Sequenced Package Model*. This section describes that model. In this first description, the basic concepts are introduced without regard for the specific protocol messages used to realize the semantics of the model. Later sections will

G.1 Discrete Package Model

An ICE subscription consists of a discrete set of packages delivered, in order, over a period of time. Consider the following diagram representing the delivery of individual packages, each labeled `P` and positioned along a time-line:

```
->---|-----P-----P-------P-----P------P-------P----P-----> t
    t=0
```

ICE defines the term *collection* to mean the current content of a subscription. In the Headlines.com example discussed in 2.1.1, the collection consists of all the headline text, thumbnail images, etc., existing on a Syndicator or Subscribers site at any point in time. In the Parts Information Scenario described in 2.1.2, the collection consists of the complete set of service bulletins, price lists, etc., again as it exists at any one point in time.

ICE uses the <package as the atomic unit of collection manipulation; the only way for a Syndicator to change a Subscribers collection is for the Syndicator to send a package to the Subscriber (push or pull). It is not possible for the Syndicator to send a "naked" content file unless it is part of a package. Similarly, a Subscriber cannot request an update for an individual file; the only thing the Subscriber can do is request a new package of updates from the Syndicator.

It follows from this model that the state of a Subscribers *collection* is completely described by knowing the set of packages the Subscriber has received over time

G.2 Strictly Ordered Package Model

ICE forces a Syndicator (and a Subscriber) to view the package stream as a strictly ordered sequence of packages. This means that packages cannot be processed out of order, and all intermediate packages must be processed.

For explanatory purposes, assume for the moment that packages were numbered P1 for the first package, P2 for the second, etc., In this case the strictly ordered package model of ICE requires that the Subscriber always process package PN-1 before processing package PN.

This model may seem at first glance to be a poor match for some types of syndication, where intermediate updates might not be important. For example, in the Headlines.com example, if a Subscriber misses 10 days of headlines, it might be perfectly reasonable for the Subscriber to simply get the current set of headlines and ignore the intervening packages. The ICE model does, in fact, allow for this type of Syndication; this will be explained in a moment.

G.3 Subscription State

Given that ICE defines a package as the atomic unit of collection manipulation, and given that ICE forces a Subscriber to process all packages in a strict order, it is possible for a Syndicator (or Subscriber) to completely describe the state of the Subscribers collection with a single value: namely, an identifier indicating the position of the Subscriber within the ordered sequence of packages.

Thus, if packages were numbered with integers, consider the following package sequence:

```
->---|-----P1----P2------P3----P4-----P5-----P6----P7-----> t
     t=0
```

In this example, simply knowing the number of the last package successfully processed by a Subscriber will suffice to know the complete state of the Subscribers collection. For example, knowing that the Subscriber is "in state 5", meaning, has received and correctly processed package number 5, implies that the Subscribers collection is in the state that would be achieved by starting in an empty state, and processing packages 1 through 5, in order. Thus, a simple number by itself, e.g., "5", suffices for describing the state of the Subscribers collection.

In ICE, this "number" is called a `package sequence identifier`, and is actually not a number at all, but rather an opaque string.

A subscription state identifier is an opaque string, generated by a Syndicator, representing the state at the boundary (before or after) of package processing. Each package sent by a Syndicator to a Subscriber has two package sequence identifiers attached to it: an "old" state value representing the required state before processing the package, and a "new" state value representing the resulting state after processing the package.

Note that the identifier is completely opaque to the Subscriber. This gives the ICE implementation on the Syndicator the complete flexibility to use an implementation specific method for encoding state into this identifier. For example, the implementation might use integers as described above, or it might use timestamps, or it might use a unique key into a proprietary database as the state encoding mechanism. All of these methods are permitted, and the opaqueness of the identifier guarantees that (properly-implemented) Subscribers will not be affected by these choices.

ICE defines three distinguished values for subscription state identifiers:

- **Zero-length string**
  This is an illegal Package Sequence Identifier string. All ICE implementations MUST reject any use of this string with a 411 (Unrecognized package sequence state) status code.
- **ICE-INITIAL**
  This special string means the "null", or empty, state, and describes the state of a collection when a Subscriber first establishes a subscription (i.e., has not yet received any packages).
- **ICE-ANY**
  This special string means "any state, whatsoever", and has special semantics when used in the "old" state on a package, as defined below.

Furthermore, ICE reserves all strings beginning with **`ICE-`** (capital I, capital C, capital E, hyphen) as values for subscription states. All other values of an `old-state=` and `new-state=` are controlled by the Syndicator and are completely opaque to the Subscriber.

The requirements for Subscribers regarding Package Sequence Identifiers are:

Subscribers MAY compare two Package Sequence Identifier strings for equality; to do so, Subscribers MUST compare the entire string using an opaque character equality comparison.

Subscribers MUST NOT assume any ordering semantics regarding unequal Package Sequence Identifier strings. In particular, Subscribers MUST NOT assume that lexigraphical or alphabetical ordering has any semantic significance whatsoever. For example, Syndicators might be using simple integer strings as Sequence Identifiers, and "42" might sort earlier than "9".

Subscribers MUST interpret the three special cases (zero-length, `ICE-INITIAL, ICE-ANY`) as described above.

G.4 Packages and Package Sequence Identifiers

When a Syndicator delivers a package to a Subscriber, whether by push or pull, the package contains two sequence identifiers: the `old-state=`, which represents the state the Subscriber must be in before applying the package, and the `new-state=`, which represents the state the Subscriber will be in after applying the package.

Assume, for example, that a Syndicator is using the names of people as the Package Sequence Identifier. Using this method, a set of packages delivered over time might consist of:

```
First Package: old-state: ICE-INITIAL  new-state: STEVE
Next Package:  old-state: STEVE        new-state: GREG
Next Package:  old-state: GREG         new-state: ROGER
```

As will be shown in more detail later, a Subscriber is required to store its current Package Sequence state at all times. When it first starts a new subscription, the Subscriber starts in state `ICE-INITIAL`. In the above example, the first package the Subscriber receives must have an old-state of `ICE-INITIAL` (or `ICE-ANY`, which will be discussed next). If, due to some operational error, the Subscriber were to receive the wrong package, e.g., one that said old-state: `GREG` instead of old-state: `ICE-INITIAL`, then the Subscriber would know not to process that package and to raise an error condition.

The above model works well for subscriptions requiring a strict, fully reliable, replication of state from a Syndicator to a Subscriber. The Package Sequence model strictly forces the Subscriber to receive all packages in their proper order, and process them each individually. The protocol does this by requiring the Subscriber to remember its current Package Sequence Identifier, and to send that Identifier to the Syndicator when requesting a package update (for pull; push subscriptions are slightly more complex and will be discussed later). Thus, the Syndicator always knows what state the Subscriber is in, and the Syndicator can thus always compute what the "right" next package to send to the Subscriber.

Some models of subscriptions do not require the rigor of this model. As mentioned, the Headlines.com model can be implemented in a much simpler fashion: each package is actually a full update of the Subscriber, and there are no dependencies on intervening packages. The ICE Package Sequence model accommodates this type of subscription using the `ICE-ANY` value. When `ICE-ANY` appears in the "old-state" of a package, it means that the package can be applied by a Subscriber regardless of what state the Subscriber is in.

By using combinations of `ICE-ANY` preconditions and specific preconditions, a Syndicator can also implement hybrid models where some packages are useful regardless of the Subscribers current state.

G.5 Sequenced Package Example

An example will help tie this all together. To understand the example, assume for the moment that packages can contain files, and that they can also contain "remove" operations that refer to files delivered in previous packages. As will be explained later, packages can indeed contain these types of things, albeit in a much more general (and complex) way (because packages are not limited to operating only on files).

A Syndicator provides a restaurant review service; Subscribers receive updates with new restaurants, new information about existing restaurants, etc.

Reviews are stored as flat HTML files in a URL structure that looks like:

```
/restaurants/<<<restaurant-name>>>.html
```

In other words, when a new restaurant is reviewed, the Subscriber simply receives a package with the new HTML file. When an existing review is updated, the Subscriber receives a package with an update for an HTML file. `A /restaurants/index.html` file is sent each time to provide navigation

When a restaurant burns down to the ground (it's a rough town), the Syndicator makes certain to include a remove operation in the next package update.

Assume for the moment that the service is just starting up and there is only one Subscriber. The service is launched with only 3 restaurant reviews. The package stream generated over time by the Syndicator might look something like this:

```
Time = 1 -- package P1:
   add /restaurants/bobs.html
   add /restaurants/joes.html
   add /restaurants/moms.html
   add /restaurants/index.html
Time = 2 -- package P2:
   comment: a new restaurant opened, and bob's is updated
   add    /restaurants/anns.html
   update /restaurants/bobs.html
   update /restaurants/index.html
Time = 3 -- package P3:
   comment: someone burned mom's place down
   remove /restaurants/moms.html
   update /restaurants/index.html
```

At this point assume that a new Subscriber signs up. That Subscriber needs all three packages P1, P2, P3, in that order. The Syndicator will know this because the Syndicator (by definition) knows that it is currently in state "P3", and it will know that the Subscriber is in state **`ICE-INITIAL`** when the Subscriber requests its first update.

Note that, as an implementation optimization, the Syndicator can construct a special "catch up" package in this case. That would look like this:

```
add /restaurants/anns.html
add /restaurants/joes.html
add /restaurants/bobs.html
add /restaurants/index.html
```

A Syndicator implementation that does that might be more efficient than sending all three incremental updates. But whether or not this should be done is a quality-of-implementation decision made by the Syndicator. Nothing in the sequenced package model dictates one approach or the other.

Finally, assume one more package needs to get sent, this time to two Subscribers:

```
Time = 4 -- package P4:
   comment: mom's rebuilt, and there's another update for
bob's
   add    /restaurants/moms.html
   update /restaurants/bobs.html
   update /restaurants/index.html
```

As mentioned before, the Subscriber must keep track of the sequence identifier of the last successfully processed package. The Subscriber sends this sequence identifier back to the Syndicator when requesting an update, so that the Syndicator can understand the Subscribers state. The Syndicator contains the logic to understand what to do based on the Subscribers (stated) sequence identifier. In the case of an unreliable update model, the Syndicator can basically ignore the sequence identifier and just send the current package (with an old-state of **`ICE-ANY`**). In other models, the Syndicator can compute what to send by decoding the sequence identifier (which it generated in an earlier package) and using that to determine what to send.

G.6 Example Pseudo-protocol Exchange

This shows the messages exchanged in the above example when the new Subscriber was added between Time 3 and Time 4 in the above sequence.

```
SUB ==> SYN I'm subscribing to RESTAURANTS
SYN ==> SUB OK

SUB ==> SYN GetPackage, my state is ICE-INITIAL
SYN ==> SUB

- three packages
P1, old-state: ICE-INITIAL new-state: XYZ-1
P2, old-state: XYZ-1       new-state: XYZ-2
P3, old-state: XYZ-2      new-state: XYZ-3
```

Alternatively, this last message could have been:

```
SUB ==> SYN I'm subscribing to RESTAURANTS
SYN ==> SUB OK

SUB ==> SYN GetPackage, my state is ICE-INITIAL
SYN ==> SUB

- one package
Px, old-state: ICE-INITIAL new-state: XYZ-3
```

where the "Px" package would be a customized package designed specifically to get a Subscriber from the initial state to the current state. The key point is the separate specification of a list of packages to be received, and an explicit statement about what the state will be after processing the packages.

It is entirely the Syndicators discretion as to what the best way to update the Subscriber is (e.g., sending all the incremental packages or sending a special catch up package).

Suppose the Subscriber comes back before Time 4 and asks for an update:

```
SUB ==> SYN GetPackage, my state is XYZ-3
SYN ==> SUB 202 Package sequence state already current
```

Later, there are updates available:

```
SUB ==> SYN GetPackage, my state is XYZ-3
SYN ==> SUB one package: P4, oldstate XYZ-3, new XYZ-4
```

G.8 Package containment model

ICE packages contain content as a set of idempotent operations: remove and add. These operations use the addressing mechanism of a *subscription element-id* to reference and manage delivered content. The method of delivery does not affect these operations. As detailed in the Sequenced Package Model section, each package moves the subscription from an old state into a new state of the subscription.

An ICE `<icedel:package` describes a set of content operations: removals (`<icedel:remove-item`) and additions (`<icedel:add).`

The content model of the ICE `< icedel:package` element is constructed so that it MUST contain some operation; at a minimum, a single removal or a single addition. If there are removal operations, they MUST be specified and, therefore, performed before any additions. It is possible that an ICE `< icedel:package` only contains removal operations. Alternatively, a `< icedel:package` may consist entirely of additions. The `<icedel:package` specifies an `old-state` and a `new-state`. Before the new-state can be reached, all of the operations contained within a package MUST be processed, and, if constraints are specified, the constraints MUST be met as well. If an operation can not be performed successfully, all previously performed operations specified in the package MUST be undone, so the Subscriber is not left in an inconsistent state with regards to the package sequence, and a surprise `< icemes:status-code` message MUST be delivered to the Syndicator indicating the type of error that occurred, such as 420 (Constraint failure). All of the operations are idempotent, i.e., it is not an error if the same content is added more than once, nor is it an error if a remove operation does not find the element to remove. In both cases the results are the same (an add operation resulted in the content existing on the Subscribers system, and a remove operation resulted in the content not existing).