ICE: Information and Content Exchange Protocol

Guidelines to Extending the ICE Protocol

Version 2.0

2004 08 01

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

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

Status of this Document

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

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

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

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

Abstract

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

Table of Contents

1. Extending the ICE Protocol

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.

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.

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

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 2.2.1 and 2.2.2.

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.

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.

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 Figure2.1, you can see where the ICE header may be extended. In the positions where `#wildCard` appears, elements from any namespace can be included.

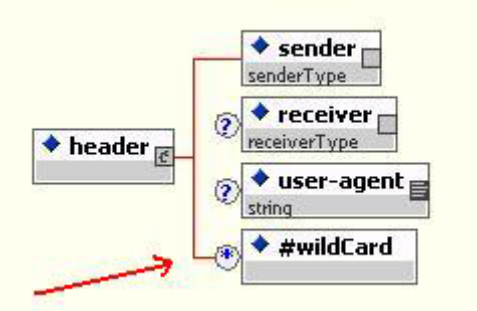

Figure2.1 ICE Extensions Allowed at Limited Points

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

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`.

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

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>
```

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.

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.

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.

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 2.1. Note that extensions may be made wherever `#wildcard` is shown.
The ICE status code is shown in Figure 3.1. Notice that we can add our own extension elements to the status code.

Figure 3.1 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>
```

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 3.2 can be extended directly or within a `<icedel:group`, `<icedel:add`, or `<icedel:remove-item`.

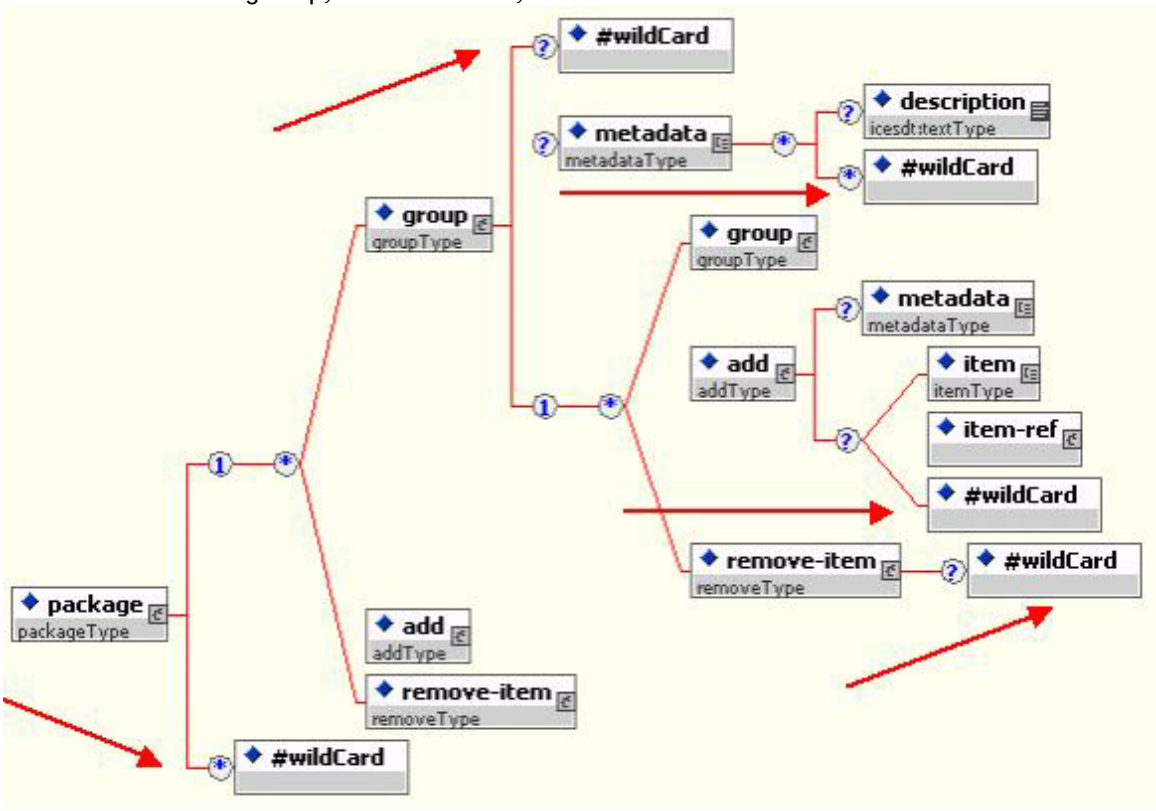

Figure 3.2 Extensions May be Made at Several Locations within the Package Elements

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.

3.3 Extensions in ICE Subscribe

Elements within the ICE subscribe namespace have also been designed to allow for controlled extensions. See Figure 3.3 to find extension nodes.

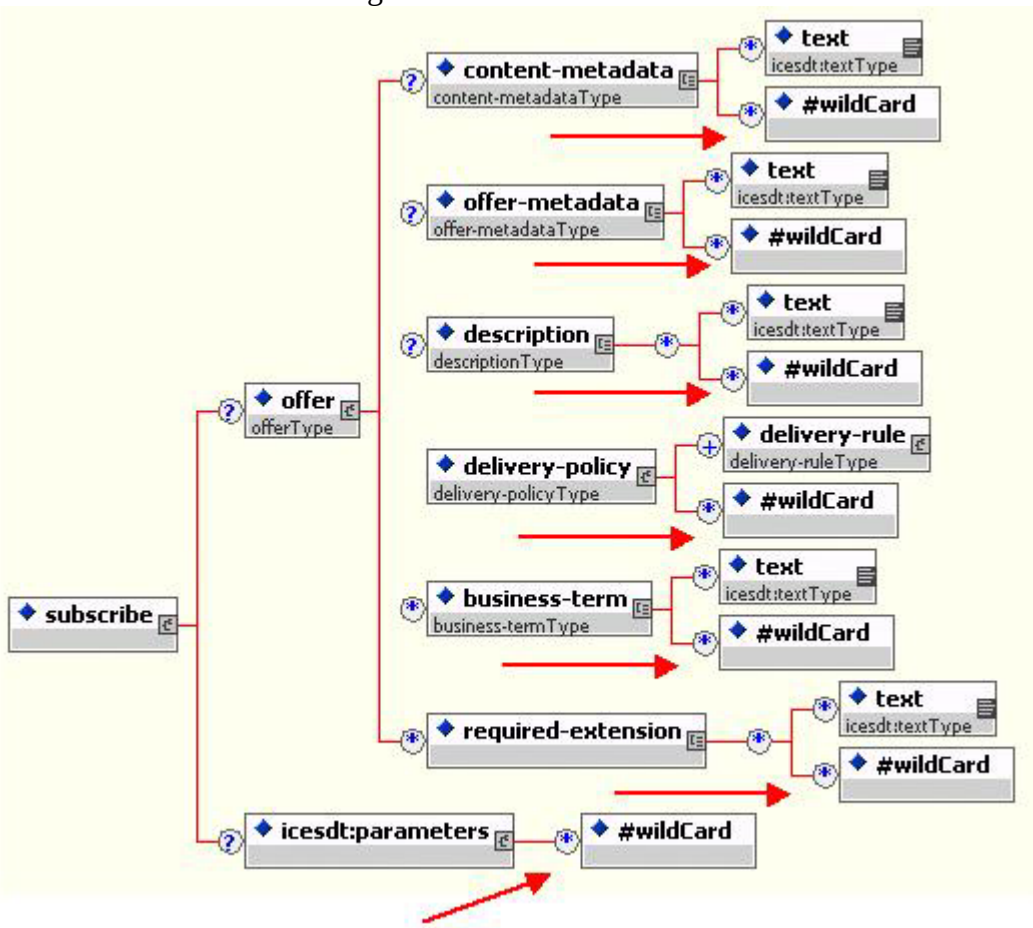

Figure 3.3 Extensions to the ICE Subscription Model

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>
```

4. Indicating ICE Extensions

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

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 4.1.

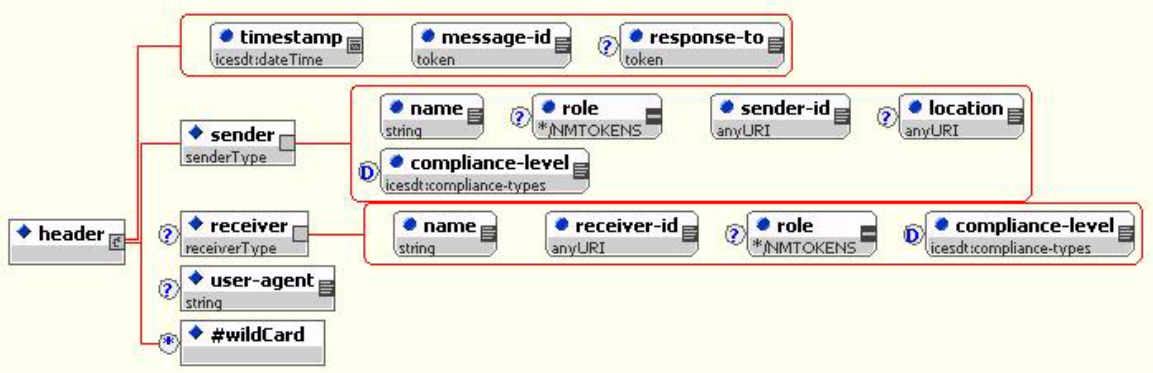

Figure 4.1 Compliance Level Attribute in ICE Message Header

4.2 ICE Offer

Extended ICE is indicated in the offer using the <icesub:required-extensions element. See Figure 4.2.

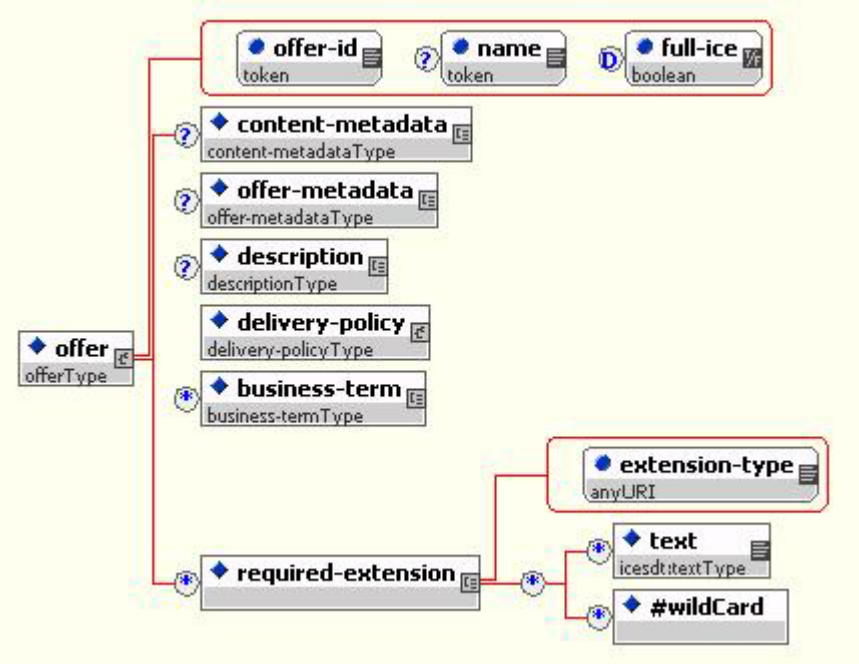

Figure 4.2 Required Extensions Indicated in Offer

> 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`".

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. 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.