



White Paper
Using ICE as a Transport Protocol
for PRISM Aggregator Messaging

Draft A
March 9, 2006

By Dianne Kennedy
IDEAlliance PRISM Program Manager

Table of Contents

1	Introduction to ICE	1
2	Introduction to PAM	1
3	What PAM Provides.....	1
4	What PAM Does Not Provide.....	2
5	ICE as a Transport for PRISM Aggregator Messages.....	2
5.1	Recommended ICE Implementation for PRISM.....	2
5.2	Recommended Subset of ICE.....	3
5.2.1	Package Attributes.....	3
5.2.3	Add.....	4
5.2.4	Remove-Item	5
5.3	Confirmation	6
6	Understanding the ICE Package Model	8
6.1	Discrete Package Model.....	8
6.2	Strictly Ordered Package Model	8
6.3	Package Sequence Identifier.....	8
6.4	Packages and Package Sequence Identifiers	10
6.5	Sequenced Package Confirmation	11
6.6	Errors in Package Delivery	11
6.7	Incremental Update of a Sequenced Package Example	11
6.7.1	Step 1: Initial Content Delivery.....	12
6.7.2	Step 2: Add additional Content.....	12
6.7.3	Step 3: Update, Add and Remove Content.....	13
7	Issues.....	14
7.1	ICE XSD / PAM DTD.....	14
7.2	PAM Status Values	14

1 Introduction to ICE

Reusing and redistributing information and content from party 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. PRISM and PAM encoding for articles are clearly examples.

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 such as PRISM, ICE provides a complete solution for syndicating any type of information between information providers and their subscribers.

Note that ICE is designed for the automated/computerized handling of content syndication management. Hence it was designed so that computers could most easily parse and act upon content.

Also note that ICE was designed to be extended. It was specifically designed so a party could add elements or attributes at key locations. To learn more see *ICE 2.0 Guidelines for Extending ICE*. We will make recommendations for certain ICE extensions to support the specific requirements of PRISM.

2 Introduction to PAM

In October 2003, PRISM WG released Version 1.0 of the PRISM Aggregator Message. This document specified a standard format for publishers to use for delivery of content to web sites and to aggregators and syndicators. PAM provides an XML DTD that facilitates a simple, flexible model for transmitting content and PRISM metadata.

The PRISM Aggregator XML tag set was designed to meet the business requirements of the members of the Working Group as they were understood in 2003. Now that the mechanism has been implemented and is in production, new requirements are causing the Working Group to investigate an alternate, syndication mechanism that will allow for enhanced control over transmission of articles from the syndicator to the aggregator.

3 What PAM Provides

PAM provides a simple message for transmitting article content and PRISM metadata. The Message is a simple envelope that carries no unique identifier. Inside the message envelope can be one or more articles. Each article carries a unique identifier

`<dc:identifier` in the header. In addition the status of the article is carried in the header. The status can be:

- A (added or new)
- C (correction to an article previously sent)
- D (delete article)
- U (updated article with corrections in place)

4 What PAM Does Not Provide

It is important to note several things that PAM does not currently provide:

- There is no identifier at the message level
- There is no mechanism to track the sequence of messages, so it is not possible to tell if article updates are in the intended sequence
- There is no mechanism to request/require acknowledgement when content is received.

ICE messaging can provide this additional functionality.

5 ICE as a Transport for PRISM Aggregator Messages

There are several possibilities to leverage ICE functionality to support PRISM. Based on my evaluation we propose to offer ICE as an alternative to the current PAM message envelope. In other words, I propose that PRISM articles are transmitted within an ICE package as an alternate to transmitting them within the simple PAM message.

5.1 Recommended ICE Implementation for PRISM

ICE was developed to manage the end-to-end syndication relationship. Whether working in a Web-services environment (Full ICE) or in a non-Web-services environment (Basic ICE), the protocol focuses on first establishing a subscription between the Syndicator (in the case of PRISM the magazine or journal) and the Subscriber (in the case of PRISM the aggregator). For the purposes of PRISM aggregation and content delivery, this level of subscription management is not required. For PRISM we will concentrate on the mechanism to deliver content and to respond with confirmations.

For the use of ICE within PRISM, we will assume that:

- The subscription will be established outside the XML messaging environment and hence ICE messages will not contain offers for syndicated content or setting delivery rules. These subscription management elements from the full ICE specification will not be initially used for PRISM.
- Package delivery and confirmation will not be implemented within the web services environment.
- The delivery of content and the confirmation of receipt will be handled by FTP push or pull or by email push. That will be agreed upon by the Syndicator and Subscriber and handled outside the XML messaging environment.

5.2 Recommended Subset of ICE

In order to make the ICE application for PRISM simple and straightforward, a simple subset of ICE has been developed to provide the functionality now sought by PRISM without over-complicating the messaging. This subset is pictured below:

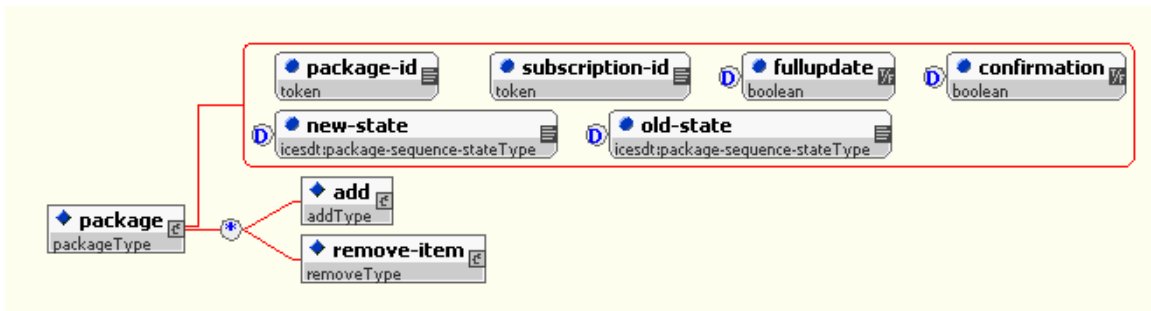


Figure 5.1 Simplified ICE Package for PRISM

The ICE package is defined within the ICE Delivery namespace (`<icedel:.` For PRISM the package will be made up of 2 elements. The ICE `<icedel:package` describes a set of content operations: additions and/or removals. See Figure 5.1. The addition process is handled by `<icedel:add` element. The operation to remove content is specified using the `<icedel:remove-item` element.

Note that sending an ICE package and its contents (articles) are within the control of a Syndicator (magazine). For PRISM, a package will most likely be all the articles in a magazine issue. The articles will be delivered within a package. If updates, additions, or removal of articles are required, that occurs based on the package.

5.2.1 Package Attributes

There are several attributes on `<icedel:package`: These attributes work together to facilitate update of a package (adding new content, removing content and updating content) as well as enabling the request for confirmation of delivery.

- **confirmation**
Default. This attribute specifies whether confirmation of receipt is required. The values are Boolean, “true” or “false.” The default is “false”.

- **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.” If full update is selected, the package that is sent overwrites the previous package with the same package ID. If partial update is selected, then the package content is incrementally updated.
- **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”.
- **subscription-id**
Required. The subscription-id is the unique id of the content feed. The Syndicator (magazine) assigns the subscription-id. For PRISM, best practice is to use the magazine title and year as the subscription-id.
- **package-id**
Required. This attribute specifies the unique id of the package within a subscription for which delivery is confirmed. For PRISM, best practice is to use identifier for the magazine issue for which articles have been sent as the package-id.

5.2.3 Add

The `<icedel:add` element is used to add new content. The `<icedel:add` enables content to be directly included in the message by using the `<icedel:item` element. In the case of PRISM, an article will be sent inside each item. The structure of `<icedel:add` for PRISM is shown in Figure 5.2.

Note that `<icedel:add` has a required attribute for the PRISM implementation that is `subscription-element-id`. This is the unique identifier at the add level and is used to track what is added and what is removed with `<icedel:remove-item`. For ease, we recommend that the subscription element ID is a duplicate the identifier carried by `<dc:identifier` within the heading of the article at the add `subscription-element-id` level.

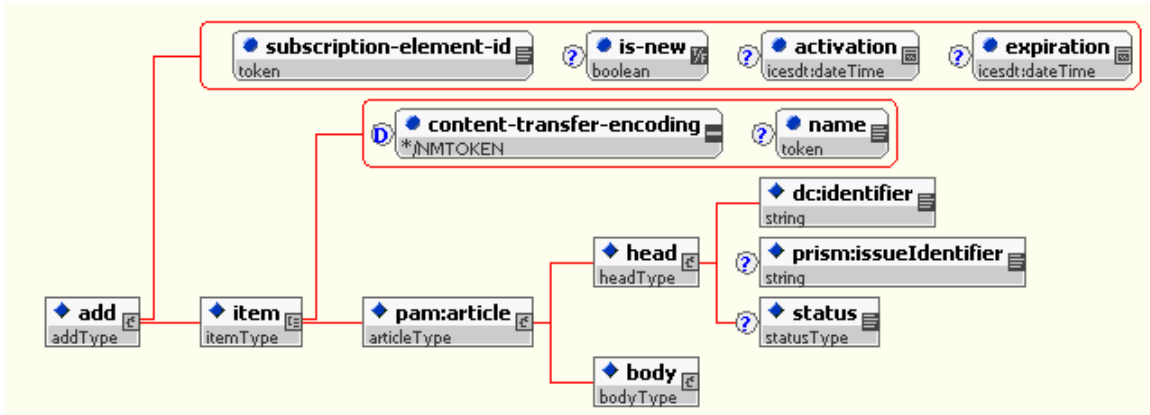


Figure 5.2 Add Element Structure

The `<icedel:item` element directly carries content from the Syndicator (magazine) to the Subscriber (journal). In the case of PRISM, the item will carry the `<pam:article`.

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

- subscription-element-id**
Required. This attribute specifies the persistent identifier of an element of a subscription that is being added. This applies to the contained item content. For PRISM, best practice is to use the `<dc:identifier` for the article.
- is-new**
Optional. This attribute specifies that the content is new. The values are Boolean, “true” or “false.” PRISM allows for an update of the content of the article (U) or to communicate a change to be made on the receiving side. ICE only supports “update” and has no mechanism to communicate a change in an article without sending the changed article. To use ICE for PRISM we recommend eliminating the “C” status that exists for PAM today.
- activation**
Optional. This attribute specifies when the addition for content is activated. The value is in the `icesdt:dateTime` format. There has not been a requirement for this feature specified by the PRISM WG, but they may find this feature useful so it has been retained in the PRISM subset of ICE.
- 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. There has not been a requirement for this feature specified by the PRISM WG, but they may find this feature useful so it has been retained in the PRISM subset of ICE.

5.2.4 Remove-Item

The `<icedel:remove-item` element is used to remove content of the subscription. In the case of PRISM this is used to withdraw an article previously sent to the Aggregator/Subscriber.

The structure of `<icedel:remove-item>` is shown in Figure 5.3.



Figure 5.3 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 that is to be removed. This matches with the `subscription-element-id` that was used when the item/article was added with `<icedel:add>`. For PRISM, best practice would be to match the `<dc:identifier>` in the article heading.

5.3 Confirmation

If the package that is being delivered has `confirmation="yes"` then the Subscriber (aggregator) is expected to return a package confirmation response to the Syndicator (magazine). Since the PRISM implementation of ICE is not a Web service, the message will not be sent automatically as a SOAP response. Instead the expectation is that an XML message will be sent by the Subscriber (aggregator) back to the Syndicator (magazine) using a mutually agreed-upon method such as FTP or email. The confirmation message structure is shown in Figure 5.4.



Figure 5.4 Confirmation structure

A sample confirmation message is shown below:

```
<icedel:confirmation confirmed="yes" subscription-id="PeopleMag06" package-id="People021106" current-state="03"/>
```

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

- confirmed**
Required. This attribute specifies whether the package delivery is confirmed. The value is Boolean (“true” or “false” and there is no default.
- subscription-id**
Required. The `subscription-id` is the unique id of the content feed. The Syndicator (magazine) assigns the `subscription-id`. For PRISM, best practice is to use the magazine title and year as the `subscription-id`.
- package-id**
Required. This attribute specifies the unique id of the package within a subscription for which delivery is confirmed. For PRISM the `package-id` will

most likely be an identifier for the magazine issue for which articles have been sent.

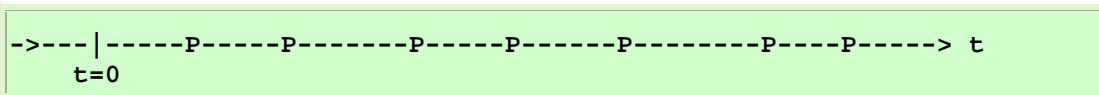
- **current-state**
Optional. The sequence identifier for package confirmation. Represents the current state of the subscription. If only a single version of the package is sent this attribute is optional.

6 Understanding the ICE Package Model

Package delivery in 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 describe the specific messages.

6.1 Discrete Package Model

In ICE a discrete set of packages may be delivered, in order, over a period of time. For PRISM, packages will likely be articles in a particular issue of a particular magazine. Consider the following diagram representing the delivery of individual packages, each labeled P and positioned along a time-line:



ICE defines the term *collection* to mean the current state of content sent from the Syndicator (magazine) to the Subscriber (aggregator) at any point in time.

ICE uses the package as the atomic unit of collection manipulation; the only way for a Syndicator to change a Subscriber's collection is for the Syndicator to send a package to the Subscriber. It is not possible for the Syndicator to send a "naked" file such as an article 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 Subscriber's *collection* is completely described by knowing the set of packages the Subscriber has received over time.

6.2 Strictly Ordered Package Model

ICE forces a Syndicator (magazine) 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 P_1 for the first package, P_2 for the second, etc.,. In this case the strictly ordered package model of ICE requires that the Subscriber always process package P_1 before processing package P_2 etc..

6.3 Package Sequence Identifier

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 (magazine) to completely describe the state of the Subscriber's 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 per the following definition:

Definition: A *Package Sequence 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 a distinguished values for the initial Package Sequence Identifier:

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).

All other values of a Package Sequence Identifier are controlled by the Syndicator (magazine) and are completely opaque to the Subscriber.

The requirements for Subscribers (aggregators) 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 lexicographical 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 `ICE-INITIAL` as the first package delivery identifier.

6.4 Packages and Package Sequence Identifiers

When a Syndicator (magazine) delivers a package to a Subscriber (aggregator) the package contains a set of two sequence identifiers: the *old-state*, which represents the state the Subscriber (aggregator) 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 (magazine) 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
Second Package	old-state: STEVE	new-state: GREG
Next Package	old-state: GREG	new-state: BETTY

For ICE in an automated content delivery environment, a Subscriber (aggregator) is required to store its current Package Sequence state at all times. When it first starts a new subscription, the Subscriber (magazine) 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.

Because the PRISM ICE Implementation is tailored for a non-automated content delivery environment, we are specifying that as a best practice, PRISM package sequence identifiers will be ordered integers. We will adopt the practice of using `P#` to identify each package after the first content package is delivered. Thus for the simple PRISM implementation of ICE we expect:

First Package	old-state: ICE-INITIAL	new-state: P1
Second Package	old-state: P1	new-state: P2
Next Package	old-state: P2	new-state: P3

6.5 Sequenced Package Confirmation

Not only are the package sequence ids used to determine the order in which packages are to be processed in, but they are used to confirm package receipt.

Suppose the Syndicator (magazine) delivered a package that contained updates, adds and removals for a particular issue of a magazine with the message:

```
<package subscription-id="People06"
  package-id="People021506" confirm="yes" old-state="P2"
  new-state="P3"> </package>
```

This message requires a confirmation response from the Subscriber (aggregator). That message would not only contain the confirmation but must reference it to the specific package in a delivery sequence:

```
<confirmation subscription-id="People06"
  package-id="People021506" current-state="P3"/>
```

6.6 Errors in Package Delivery

The confirmation message is a mechanism whereby the Subscriber (aggregator) is required to acknowledge the receipt of a package. But what happens if a package is not received or acknowledged and the delivery sequence is corrupted. Because for PRISM the use of ICE is not intended to be automated, communications between the Syndicator (magazine) and the Subscriber (aggregator) may need to take place by email or telephone if the package delivery sequence needs repair. The good news is that by employing this simple subset of ICE, the Subscriber (aggregator) will know that they are missing content deliveries for a package because they can determine that the **old-state / new-state** sequence identifiers are out of order.

6.7 Incremental Update of a Sequenced Package Example

This example is provided to help tie this all together. To understand the example, assume for the moment that a package of PRISM content will contain all articles that make up an issue of a magazine. Also assume that several deliveries with the same package-id (the same magazine and issue) are sent to add new articles, update some articles and remove others that were delivered in previous packages.

6.7.1 Step 1: Initial Content Delivery

A Syndicator (magazine) provides an initial delivery to the Subscriber (aggregator) of 3 articles for People Magazine February 15, 2005 issue:

```
<package subscription-id="People06"
  package-id="People021506" confirm="yes" old-
state="ICE-INITIAL"      new-state="P1">
  <add subscription-element-id = "p001" is-new="true">
    <item><article> ... </article></item>
  </add>
  <add subscription-element-id = "p002" is-new="true">
    <item><article> ... </article></item>
  </add>
  <add subscription-element-id = "p003" is-new="true">
    <item><article> ... </article></item>
  </add>
</package>
```

Since the Syndicator (magazine) has asked for confirmation of delivery, the Subscriber (aggregator) must reply with the message:

```
<confirmation subscription-id="People06"
  package-id="People021506" current-state="P1"/>
```

6.7.2 Step 2: Add additional Content

The Syndicator (magazine) adds two new articles to the package. Note that since we are just adding content to the previous package sent, we have specified `fullupdate="false"`. Also note that we have a new state for this package.

```
<package subscription-id="People06"
  package-id="People021506" confirm="yes"
fullupdate="false" old-state="P1" new-state="P2">
  <add subscription-element-id = "p004" is-new="true">
    <item><article> ... </article></item>
  </add>
  <add subscription-element-id = "p005" is-new="true">
    <item><article> ... </article></item>
  </add>
</package>
```

Since the Syndicator (magazine) has asked for confirmation of delivery, the Subscriber (aggregator) must reply with the message:

```
<confirmation subscription-id="People06"
  package-id="People021506" current-state="P2"/>
```

6.7.3 Step 3: Update, Add and Remove Content

The Syndicator magazine now adds another article, updates the content of an article previously sent and removes an article previously sent. Note that when using ICE we update articles, but will not have the mechanism to send corrections unless we extend ICE to do so!

```
<package subscription-id="People06"
  package-id="People021506" confirm="yes" old-state="P2"
  new-state="P3">
  <add subscription-element-id = "p006" is-new="true">
    <item><article> ... </article></item>
  </add>
  <add subscription-element-id = "p002" is-new="false">
    <item><article> ... </article></item>
  </add>
  <remove-item subscription-element-id = "p004" />
</package>
```

Since the Syndicator (magazine) has asked for confirmation of delivery, the Subscriber (aggregator) must reply with the message:

```
<confirmation subscription-id="People06"
  package-id="People021506" current-state="P3"/>
```

7 Issues

Several issues have been identified as this recommendation to use a subset of ICE to facilitate the managed delivery of PAM messages. These issues are documented here and must be discussed and resolved before a final implementation is documented.

7.1 ICE XSD / PAM DTD

While ICE 1.0 was specified using an XML DTD in 1998, ICE 2.0 made the transition to become a Web Service and hence is specified as an XSD. Because it is not possible to mix/match XML tags specified by a DTD and those specified as an XSD, there is a basic incompatibility between PAM/PRISM and ICE 2.0.

For the purpose of this evaluation, a simple XSD was created for PAM. This was accomplished by exporting an XSD from the pam-norm.dtd using Tibco Turbo XML.

In order to identify ICE as an alternate managed delivery mechanism for PAM articles, we must either create a subset of ICE 1.1 DTD for use with pam-norm.dtd or we will finally need to develop an XSD for the PAM article.

7.2 PAM Status Values

Currently PAM has status values of

A (added or new)

C (correction to an article previously sent)

D (delete article)

U (updated article with corrections in place)

ICE on the other hand has no concept of sending corrections. Rather because ICE was designed to work in a highly automated environment, it expects that the Syndicator will send updated or corrected content but not corrections to be applied on the Subscriber (aggregator) side.

A decision needs to be made whether we can eliminate the “C” functionality of PAM when articles are being delivered via an ICE implementation or whether we should extend ICE to allow for corrections to be sent.