# Simple Sharing Extensions for Atom and RSS

Version: 0.93

Contact: Jack Ozzie, George Moromisato, Matt Augustine, Paresh Suthar and Steven Lees, Microsoft Corporation

Updated: 05/07/2007

Change log:

0.93 –May 7, 2007

* Removed OPML binding
* Fixed issues identified by feedvalidator.org
* Updated the sync algorithms to improve efficiency and accuracy
* Removed version attribute from sx:sharing, made sx:sharing optional
* Changed date format to RFC 3339 for all date-time values
* Added Atom binding/samples
* Added section to describe since, until and expires for complete and partial feeds

0.91 - Original

### 1 Overview

The scope of Simple Sharing Extensions (SSE) is to define the *minimum* extensions necessary to enable loosely-cooperating applications to use XML-based container formats such as Atom and RSS as the basis for *item sharing* – that is, the bi-directional, asynchronous synchronization of new and changed items amongst two or more cross-subscribed feeds.

Simple Sharing extends the following specifications:

Atom 1.0 - <http://www.ietf.org/rfc/rfc4287>

RSS 2.0 - <http://blogs.law.harvard.edu/tech/rss>

One of the guiding principles of SSE is to reinvent as little as possible—hence the use of Atom and RSS in this spec as the underlying format for exchanging SSE data. It is expected that there will be additional container format bindings for SSE in the future.

##### 1.1 Namespaces and Version

The XML namespace URI for the XML data format described in this specification is:

http://www.microsoft.com/schemas/sse

In this spec, the prefix "sx:" is used for the namespace URI identified above.

Atom example:

<feed xmlns="http://www.w3.org/2005/Atom"

xmlns:sx="http://www.microsoft.com/schemas/sse">

RSS example:

<rss version=2.0 xmlns:sx="http://www.microsoft.com/schemas/sse">

##### 1.2 Notational Conventions

1. In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119.
2. The term *item* denotes a typed data entity and the basic unit of sharing and synchronization.
3. The term *endpoint* denotes an entity that participates in the synchronization of shared items with other endpoints. An endpoint can act as a publisher, a subscriber or both.
4. An endpoint's *item set* is a complete set of related items as determined by the endpoint.
5. A *feed* is an addressable collection of items in an endpoint's item set represented in the Atom 1.0 or RSS 2.0 format. The feed can be partial (only the items that have changed within a given time window) or complete (all of the items in the endpoint's item set are contained in the feed as of its publishing).
6. A *subscription* is a unidirectional relationship between two endpoints where one endpoint acting as a subscriber pulls feeds from the other endpoint, which acts as a publisher.
7. The term *updated* is used to include items created, changed or deleted.
8. The term *incorporate* denotes the act of creating or updating a local feed item reflecting those updates made to the item in a remotely subscribed feed.
9. The term *unpublish* denotes exclusion from publication of one or more items that have previously existed in a published feed.
10. All endpoint identifiers MUST conform to the syntax for Namespace Specific Strings (the NSS portion of a URN) in RFC 2141.
11. Unless otherwise specified, if an xml element attribute value is set, it MUST be set to a non-empty string value.
12. Unless otherwise specified, string attributes are unbounded. An implementation MAY prevent incorporation of an item if one or more string attributes exceed bounds imposed by that implementation.
13. Unless otherwise specified, the comparison of string attributes MUST be done using the Unicode values of each character and be culture-insensitive. Unicode values can be represented as "U+xxxx" where xxxx is the hexadecimal code point of the character. A string starting with "U+xxxx" comes before a string starting with "U+yyyy", if xxxx is less than yyyy.
14. All date-time values MUST be conform to the Date and Time specification of RFC 3339.
15. Unless otherwise specified, the comparison of date-time attributes MUST be done using normalized values as defined by the convention above and MUST NOT be done using string comparison.

##### 1.3 Usage Model

Imagine two loosely coupled endpoints, A and B, that wish to share and co-edit a set of independent items in an Atom feed. The two endpoints can use SSE to synchronize the set. The process would look like this:

1. Endpoint **A** maintains an item set. Endpoint **A** *publishes* the set as a feed conforming to SSE format. Let's call this **feed-A**.
2. Endpoint **B** also maintains an item set. Endpoint **B** *subscribes* to **feed-A** and incorporates the items into its own set.
3. Endpoint **B** *publishes* its own set (including items it got from **A**) as a feed conforming to SSE. Let's call this **feed-B**.
4. Endpoint **A** *subscribes* to **feed-B** and *incorporates* items from endpoint **B** into its own set. In effect, endpoints **A** and **B** mutually publish/subscribe to each other's feeds.
5. When endpoint **A** adds or changes an item, that update is reflected in **feed-A** and endpoint **B** receives the change when it reads the feed.
6. Similarly, when endpoint **B** adds or changes an item, the update is published in **feed-B** and endpoint **A** receives the change.

The extensions described in the Simple Sharing Extensions enable feed readers and publishers to generate and process incoming item changes in a manner that enables *consistency* to be achieved. In order to accomplish this, SSE introduces concepts such as per-item *change history* (to manage item versions and update conflicts) and *tombstones* (to propagate deletions, and un-deletions).

##### 1.4 Example Feeds

These feeds demonstrate a series of updates to single item.

###### 1.4.1 Atom Feed

<?xml version="1.0" encoding="utf-8"?>

<feed xmlns="http://www.w3.org/2005/Atom"

xmlns:sx="http://www.microsoft.com/schemas/sse">

<title>To Do List</title>

<subtitle>A list of items to do</subtitle>

<link rel="self" href="http://example.com/partial.xml"/>

<author>

<name>Ray Ozzie</name>

</author>

<updated>2005-05-21T11:43:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aaa</id>

<sx:sharing since="2005-02-13T18:30:02Z"

until="2005-05-23T18:30:02Z" >

<sx:related link="http://example.com/all.xml" type="complete" />

<sx:related link="http://example.com/B.xml" type="aggregated"

title="To Do List (Jacks Copy)" />

</sx:sharing>

<entry>

<title>Buy groceries</title>

<content>Get milk, eggs, butter and bread</content>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<updated>2005-05-21T11:43:33Z</updated>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="3">

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</entry>

</feed>

###### 1.4.2 RSS Feed

<?xml version="1.0" encoding="utf-8"?>

<rss version="2.0" xmlns:sx="http://www.microsoft.com/schemas/sse">

<channel>

<title>To Do List</title>

<description>A list of items to do</description>

<link> http://example.com/partial.xml </link>

<sx:sharing since="2005-02-13T18:30:02Z"

until="2005-05-23T18:30:02Z" >

<sx:related link="http://example.com/all.xml" type="complete" />

<sx:related link="http://example.com/B.xml" type="aggregated"

title="To Do List (Jacks Copy)" />

</sx:sharing>

<item>

<title>Buy groceries</title>

<description>Get milk, eggs, butter and bread</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="3">

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</item>

</channel>

</rss>

### 2 Extensions

##### 2.1 <sx:sharing> element within Atom <feed> or RSS <channel>

The sx:sharing element is optional. If it exists, the **sx:sharing** element MAY contain one or more **sx:related** elements.

|  |  |
| --- | --- |
| Attributes | Description |
| since | *An optional, string attribute.* If present, its value is set by the publisher as a lower bound of items contained within the feed. See section 5 for more information.  NOTE: If this attribute is exists, the **until** attribute MUST also exist. |
| until | *An optional, string attribute*. If present, its value is set by the publisher as an upper bound of items contained within the feed. See section 5 for more information.  NOTE: If this attribute is exists, the **since** attribute MUST also exist. |
| expires | *An optional, date-time attribute.* If present, represents a publisher suggested date-time before which subscribers SHOULD read the feed in order to avoid missing item updates. See section 5 for more information.  The value for this attribute SHOULD be interpreted as a best effort, uncalibrated value. |

##### 2.2 <sx:related> element within <sx:sharing>

The **sx:related** element is optional, but when present contains information about related feeds or locations.

|  |  |
| --- | --- |
| Attributes | Description |
| link | *A required, URL attribute.* The URL for related feed or location. |
| title | *An optional, string attribute.* If present, represents the name or description of the related feed. |
| type | *A required, string attribute.* This attribute can have one of the following values:  **"complete"** if the link points to a feed that contains the complete collection of items for this feed. **"aggregated"** if the link points to a feed whose contents are being incorporated into this feed by the publisher. |

###### 2.2.1 Aggregated Feeds

In the case where a publisher’s feed has incorporated items from other feeds, it can be useful for subscribers to see more detailed information about the other feeds. In the case of feed sharing as envisioned by this specification, this feature can also be used to notify subscribing feeds of the feeds of other participants which they might also wish to subscribe to.

###### Atom Example

<feed>

<sx:sharing>

<sx:related link="http://example.com/all.xml" type="complete" />

<sx:related link="http://example.com/B.xml" type="aggregated"

title="To Do List (Jacks Copy)" />

</sx:sharing>

...

</feed>

###### RSS Example

<channel>

<sx:sharing>

<sx:related link="http://example.com/all.xml" type="complete" />

<sx:related link="http://example.com/B.xml" type="aggregated"

title="To Do List (Jacks Copy)" />

</sx:sharing>

...

</channel>

##### 2.3 <sx:sync> element within feed item elements (e.g. <entry> in Atom or <item> in RSS)

The most important extension described in this specification is the **sx:sync** element, which contains the information required for synchronization. This element is a child of an element (e.g. <entry> or <item>) that is a child of the **feed** or **channel** element. This is a REQUIRED element of all items in all feeds wishing to participate in SSE-based synchronization.

|  |  |
| --- | --- |
| Attributes | Description |
| id | *A required, string attribute.* This is the identifier for the item. The ID MUST be globally unique within the feed and it MUST be identical *across* feeds if an item is being shared or synchronized as part of multiple distinct independent feeds.  Note that Atom has a similar requirement for each entry to have a unique id. While the Atom entry id could be used for the sync id at the publisher’s discretion, implementers MUST NOT assume that the Atom id for the entry matches the sync id. Likewise, if the RSS item includes a GUID, implementers MUST NOT assume that the GUID is the same as the sync id.  NOTE: The ID is assigned by the creator of the item, and MUST NOT be changed by subsequent publishers. Applications will collate and compare these identifiers; therefore they MUST conform to the syntax for Namespace Specific Strings (the NSS portion of a URN) in RFC 2141. |
| updates | *A required, integer attribute.* This is number of updates applied to an item, where valid values are from 1 to 2^31-1.  NOTE: The attribute’s value starts at 1 for new items. |
| deleted | *An optional, boolean attribute.* If present and its value is "true" (lower-case), it indicates that the item has been deleted and this is a *tombstone*. If not present, or if present with value of "false", then the item is not deleted. All other values are invalid. |
| noconflicts | *An optional, boolean attribute.* If present and its value is "true" (lower-case), it indicates that conflict preservation SHOULD NOT be performed. If not present, or present with a value of "false", then it indicates that conflict preservation MUST be performed for the item. All other values are invalid.  Note: This value MUST only be set once, and SHALL only be set when the **updates** attribute value is 1. |

##### 2.4 <sx:history> element within <sx:sync>

The **sx:sync** element MUST contain at least one **sx:history** sub-element. These sub-elements represent information about updates to the item, with the most recent update being first/topmost.

|  |  |  |
| --- | --- | --- |
| Attributes | Description | |
| sequence | *A required, integer attribute.* This is the sequencing of individual updates for the purposes of conflict detection, where valid values are from 1 to 2^31-1.  Note: The sequence number is typically assigned by copying the **updates** attribute value on **sx:sync**, after it has been incrementedat the time of an update. However, in cases where the **by** attribute is specified, an implementation can choose to assign any integer value as long as it is guaranteed to be greater than the **sequence** attribute of other **sx:history** sub-element of the **sx:sync** element with a matching **by** attribute. | |
| when | *An optional, date-time attribute.* If present, represents the date-time for the device that performed the item modification.  The value for this attribute SHOULD be interpreted as a best effort, uncalibrated value.  Note: Either or both of the **when** or **by** attributes MUST be present; it is invalid to have neither. | |
| by | *An optional, string attribute.* If present, represents the text attribute identifying the unique endpoint that made the modification.  Note: Either or both of the **when** or **by** attributes MUST be present; it is invalid to have neither. |

##### 2.5 <sx:conflicts> element within <sx:sync>

The **sx:sync** element might contain an **sx:conflicts** sub-element. When the **sx:conflicts** element is present, it MUST contain one or more item sub-elements, where each item represents a conflicting update. See conflict preservation behavior (section 3.5) for more information.

|  |  |
| --- | --- |
| Attributes | Description |
| N/A | N/A |

##### 2.6 <sx:unpublished> element within Atom <feed> or RSS <channel>

The **feed** or **channel** element MAY contain a **sx:unpublished** sub-element. If the **sx:unpublished** element is present, it MUST contain one or more sub-elements, each of which MUST have the **sx:sync** sub-element. See unpublish behavior (section 4.1) for more information.

|  |  |
| --- | --- |
| Attributes | Description |
| N/A | N/A |

### 3 Required Behaviors for Global Consistency

To assure consistency when modifying a local feed, the following behaviors will be defined:

* + - * Creation
      * Update/Deletion
      * Conflict Resolution

To assure consistency when processing incoming (subscribed) feeds, the following behaviors will be defined:

* Merge

##### 3.1 Creation Behavior

When creating a new item as the result of a local operation, implementations MUST:

1. Create a **sx:sync** element as follows:
2. Set the **id** attribute (in conformance with section 2.3)
3. Set the **updates** attribute to “1”
4. Optionally set the **noconflicts** attribute
5. Optionally set the **deleted** attribute if the item is tombstoned from birth
6. Create a **sx:history** element as follows:
7. Set the **sequence** attribute (in conformance with section 2.4)
8. Do one or both of the following:

i. Set the **when** attribute to the current date-time. Note that this attribute is optional if the **by** attribute is set, but it is RECOMMENDED that this attribute be set whenever possible.

ii. Set the **by** attribute to the current endpoint's identifier. Note that this attribute is optional if the **when** attribute is set, but it is RECOMMENDED that this attribute be set whenever possible.

1. Add the **sx:history** element as the topmost sub-element of the **sx:sync** element.
2. Add the **sx:sync** element as a sub-element of the item’s element
3. Append the item’s element as a sub-element of the **feed** element.
4. Optionally update the **until** attribute for the **sx:sharing** element

Here is what an item could look like after being created:

###### Atom Example

<entry>

<title>Buy groceries</title>

<content>Get milk and eggs</content>

<updated>2005-05-21T09:43:33Z </updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="1">

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</entry>

###### RSS Example

<item>

<title>Buy groceries</title>

<description>Get milk and eggs</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="1">

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</item>

##### 3.2 Update/Deletion Behavior

When updating or deleting an item via a local operation, implementations MUST:

1. Update the **sx:sync** element as follows:
2. Increment the **updates** attribute by 1
3. Create a **sx:history** element as follows:
4. Set the **sequence** attribute of the **sx:history** element to one of the following:
5. The same value as the **updates** for the **sx:sync** element
6. If the **by** attribute is set, the implementation can set the value to any value as long as it is guaranteed to be greater than the **sequence** attribute of any **sx:history** sub-element (preexisting or current) of the **sx:sync** element with a matching **by** attribute.
7. Do one or more of the following:
8. Set the **when** attribute to the current date-time. Note that this attribute is optional if the **by** attribute is set, but it is RECOMMENDED that this attribute be set whenever possible.
9. Set the **by** attribute to the current endpoint's identifier. Note that this attribute is optional if the **when** attribute is set, but it is RECOMMENDED that this attribute be set whenever possible
10. Add the **sx:history** element as the topmost sub-element of the **sx:sync** element.
11. If this is a deletion, set the **deleted** attribute on the **sx:sync** element to true.
12. Optionally update the **until** attribute for the **sx:sharing** element

Additionally implementations MAY:

1. Truncate **sx:history** sub-elements, except for the topmost **sx:history** sub-element. Note that truncation of **sx:history** sub-elements that have the highest **sequence** attribute value for a given **by** attribute value, or that do not have a **by** attribute value, might harm subsequent conflict detection.
2. If this is a deletion, remove the item’s data as long as the item’s **sx:sync** and topmost **sx:history** sub-elements are preserved. Note that truncation or removal of data might cause unexpected results if a subsequent conflict occurs for the item. Also note that publishers SHOULD rely on the **window** attribute of the **sx:sharing** element to determine when to stop publishing items.

Here is what the same item in the above example would look like after having been updated in sequence by the same endpoint:

###### Atom Example#1

<entry>

<title>Buy groceries</title>

<content>Get milk, eggs and butter</content>

<updated>2005-05-21T10:43:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="2">

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</entry>

###### RSS Example#1

<item>

<title>Buy groceries</title>

<description>Get milk, eggs and butter</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="2">

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</item>

Here is what the same item in the above example would look like after having been updated in sequence by another endpoint:

###### Atom Example #2

<entry>

<title>Buy groceries</title>

<content>Get milk, eggs, butter and bread</content>

<updated>2005-05-21T11:43:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="3">

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</entry>

###### RSS Example #2

<item>

<title>Buy groceries</title>

<description>Get milk, eggs, butter and bread</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="3">

<sx:history sequence="3" when="2005-05-21T09:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</item>

##### 3.3 Merge Behavior

When a subscribing endpoint incorporates items from a publishing endpoint’s feed, these items must be merged with the existing local items. The act of merging items from an incoming feed detects new items, item updates and item conflicts and produces a merged result feed. The merging of two items with the same **id** attribute value will result in a ‘winning’ item that might have conflict items. In order to merge items, implementations MUST follow this algorithm for the two items:

1. If no local item exists with the same **id** attribute value as the incoming item, add the incoming item to the merge result feed; we are done processing the incoming item.
2. Create a collection L and populate it with the local item and the local item’s conflicts (if any exist) by using the following steps:
   1. For each item sub-element of the **sx:conflicts** element for the local item:
      1. Add the item sub-element to L
   2. If the local item has a **sx:conflicts** sub-element, remove it
   3. Add the local item to L
3. Create a collection I and populate it with the incoming item and the incoming item’s conflicts (if any exist) by using the following steps:
   1. For each item sub-element of the **sx:conflicts** element for the incoming item:
      1. Add the item sub-element to I
   2. If the incoming item has a **sx:conflicts** sub-element, remove it
   3. Add the incoming item to I
4. Create a collection M that will be used to contain items that will appear in the merged result feed
5. Create a reference W for the current ‘winning’ item and set it to an unassigned value
6. Using L as the outer collection and I as the inner collection, perform the following step
7. For each item X in outer collection:
   1. For each item Y in inner collection:
8. Determine if X is subsumed1 by Y – if so then remove X from the outer collection; process the next item in the outer collection
   1. Add X to M
   2. If W has not been assigned a value, set W to X; process the next item in the outer collection
   3. Determine if X should be declared as the new ‘winning’ item3 – if so set W to X.
9. Using I as the outer collection and L as the inner collection, perform step 7 again
10. Add W to the merge result feed
11. If the **noconflicts** attribute is set to true, then we are done processing
12. If M contains more than one item:
    1. Create a **sx:conflicts** element and add it as a sub-element of the **sx:sync** element for W
    2. For each item Z in M:
       1. If Z equals W (i.e. they are the same item), then process the next item in M
       2. Add Z as a sub-element of the **sx:conflicts** element created in step 11a.

Note that sub-elements of a **sx:conflicts** element are always preserved as a flat, unordered list.

1 Item Subsumption

An item X is subsumed by an item Y when the topmost **sx:history** sub-element of X is subsumed2 by one of the **sx:history** sub-elements of Y, where X and Y are items with the same **id** attribute value. In order to determine if X is subsumed by Y, implementations MUST perform the following:

1. Set HX as the topmost **sx:history** sub-element of X
2. For each **sx:history** sub-element HY of item Y:
   1. Compare HX with HY to see if HX is subsumed2 by HY – if so then X is subsumed by Y
3. If no determination has been made as to whether X is or is not subsumed by Y, then X is not subsumed by Y

2 History Subsumption

To determine if a **sx:history** element HX is subsumed by a **sx:history** element HY, implementations MUST perform the following comparisons, in order, for HX and HY:

1. If a **by** attribute exists for HX:
   1. If HY has the same **by** attribute value as HX, and HY has an equal or greater **sequence** attribute value than HX, then HX is subsumed by HY
2. If no **by** attribute exists for HX:
   1. If no **by** attribute exists for HY, and the **when** and **sequence** attribute values for HX respectively matches the **when** and **sequence** attribute values for HY, then HX is subsumed by HY
3. If no determination has been made as to whether HX is or is not subsumed by HY, then HX is not subsumed by HY

3 Winner Picking

The ‘winning’ item between an item X and an item Y is the item with most recent update , where X and Y are items with the same **id** attribute value. In order to determine the ‘winning’ item, implementations MUST perform the following comparisons, in order, for X and Y:

* 1. If X has a greater **updates** attribute value for the **sx:sync** sub-element than Y’s, then X is the ‘winning’ item
  2. If X has the same **updates** attribute value for the **sx:sync** sub-element as Y:
     + - 1. If X has a **when** attribute for the topmost **sx:history** sub-element and Y does not, then X is the ‘winning’ item
         2. If X has a **when** attribute value for the topmost **sx:history** sub-element and that is chronologically later than Y’s, then X is the ‘winning’ item
         3. If X has the same **when** attribute for the topmost **sx:history** sub-element as Y:
     1. If X has a **by** attribute for the topmost **sx:history** sub-element and Y does not, then X is the ‘winning’ item
     2. If X has a **by** attribute value for the topmost **sx:history** sub-element that is collates greater (see Section 2.4 for collation rules) than Y’s, then X is the ‘winning’ item
  3. Y is the ‘winning’ item

Here is what the same item in the above example would look like after having been independently updated by two endpoints, causing a conflict:

###### Atom Example

<entry>

<title>Buy groceries - DONE</title>

<content>Get milk, eggs, butter and bread</content>

<updated>2005-05-21T12:43:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="4">

<sx:history sequence="4" when="2005-05-21T12:43:33Z" by="GPM7383"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

<sx:conflicts>

<entry>

<title>Buy groceries</title>

<content>Get milk, eggs, butter and rolls</content>

<updated>2005-05-21T12:43:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="4">

<sx:history sequence="4" when="2005-05-21T12:43:33Z" by="JEO2000"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</entry>

</sx:conflicts>

</sx:sync>

</entry>

###### RSS Example

<item>

<title>Buy groceries - DONE</title>

<description>Get milk, eggs, butter and bread</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="4">

<sx:history sequence="4" when="2005-05-21T12:43:33Z" by="GPM7383"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

<sx:conflicts>

<item>

<title>Buy groceries</title>

<description>Get milk, eggs, butter and rolls</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="4">

<sx:history sequence="4" when="2005-05-21T12:03:33Z" by="JEO2000"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T010:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</item>

</sx:conflicts>

</sx:sync>

</item>

##### 3.4 Conflict Resolution Behavior

Applications can allow users to "resolve" conflicts for an item, where "resolve" means to perform an update to the item that takes the conflicting item data into account in some meaningful way. For items that have multiple conflicts, implementations MAY selectively resolve conflicts, keeping some conflicts unresolved and intact for subsequent resolution. When a particular conflict is resolved, it is removed from the list of the item’s conflicts and merged into the item’s history. In order to resolve conflicts, the following MUST occur:

1. The user is presented, within some user interface, with the ‘winning’3 item data along with the data for each of the conflicting versions of that same item (i.e. retrieved from the **sx:conflicts** sub-elements of the item). The user will then select one of the following actions:
   1. To keep the most recent data as the “resolved” state (i.e. confirming that the latest state is the desired state)
   2. To select one of the conflicting version’s data as the “resolved” state
   3. To combine or otherwise create new data as the “resolved” state (i.e. creating their own desired state)
2. The appropriate “resolved” state MUST be used to perform an update operation (section Section 3.2) to the item
3. The conflict items the user factored into their decision MUST then be merged4 into the item’s history so the user isn’t asked to resolve the same conflict more than once.

###### Note that while the steps above focus on user based conflict resolution, it does not obviate the possibility of programmatic conflict resolution. Programmatic conflict resolution would require that all endpoints execute the same conflict resolution logic for deterministic results.

4 Merging Conflict Items

In order to merge conflict items into the resolved item’s history, implementations MUST:

* + - 1. Set Y as a reference the resolved item
      2. Set Sy as a reference to the **sx:sync** sub-element for Y
      3. For each item sub-element X of the **sx:conflicts** element that has been resolved:
         1. Set SX as a reference to the **sx:sync** sub-element for X
         2. Remove X from the **sx:conflicts** element.
  1. For each **sx:history** sub-element HX of SX:

1. For each **sx:history** sub-element Hy of SY:
2. Compare HX with Hy to see if HX can be subsumed2 by HY – if so then process the next itemsub-element of X
3. Add HX as a sub-element of SY, immediately after the topmost **sx:history** sub-element of SY.

3. If the **sx:conflicts** element contains no sub-elements, the **sx:conflicts** element SHOULD be removed.

Here is what the same item in the above example would look like after conflict resolution has occurred, where one of the endpoint’s changes are picked as the resolved state:

###### Atom Example

<entry>

<title>Buy groceries - DONE</title>

<content>Get milk, eggs, butter and bread</content>

<updated>2005-05-21T12:53:33Z</updated>

<id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0aa0</id>

<author>

<name>Ray Ozzie</name>

</author>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="5">

<sx:history sequence="5" when="2005-05-21T12:53:33Z" by="GPM7383"/>

<sx:history sequence="4" when="2005-05-21T12:03:33Z" by="JEO2000"/>

<sx:history sequence="4" when="2005-05-21T12:43:33Z" by="GPM7383"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</entry>

###### RSS Example

<item>

<title>Buy groceries - DONE</title>

<description>Get milk, eggs, butter and bread</description>

<sx:sync id="item 1\_myapp\_2005-05-21T11:43:33Z" updates="5">

<sx:history sequence="5" when="2005-05-21T12:53:33Z" by="GPM7383"/>

<sx:history sequence="4" when="2005-05-21T12:03:33Z" by="JEO2000"/>

<sx:history sequence="4" when="2005-05-21T12:43:33Z" by="GPM7383"/>

<sx:history sequence="3" when="2005-05-21T11:43:33Z" by="JEO2000"/>

<sx:history sequence="2" when="2005-05-21T10:43:33Z" by="REO1750"/>

<sx:history sequence="1" when="2005-05-21T09:43:33Z" by="REO1750"/>

</sx:sync>

</item>

### 4 Optional behaviors

##### 4.1 Unpublish Behavior

The act of unpublishing an item from a feed declares a publisher’s intent that one or more items no longer meet the criteria for publication within a specific feed, where the criteria are implementation specific to the publisher. Use of **sx:unpublished** is an explicit signal used by publishers to indicate to subscribers that a set of once-published items is no longer being published. It also declares that any subsequent updates to the item MAY be ignored by the publishing endpoint upon merging unpublished items with other feeds in which the item has been modified. This is necessary because subscribers are not able to infer this information if the publisher simply stops publishing the item in the feed.

In order to unpublish an item, the following MUST occur:

1. If the **sx:unpublished** element does not exist, create it and add it as sub-element of the **feed** element.
2. Reparent the item sub-element from the **feed** element to the **sx:unpublished** element

Implementations MAY additionally perform the following:

1. Truncate or remove data for the item as long as the item’s **sx:sync** and topmost **sx:history** sub-elements are preserved.

Note that the act of unpublishing is different from the act of deletion, in that deletion affects the synchronization meta-data for the item whereas unpublishing does not. This means that, from the publishing endpoint’s perspective, unpublishing of an item is feed scoped while deletion of an item CAN span feeds in which the item is synchronized.

##### 4.2 Republish Behavior

The act of republishing an item in a feed declares the publishers intent to once again publish an item which had been previously unpublished (see Section 4.1). It also declares that any updates to the item SHOULD be applied by the publishing endpoint upon merging with other feeds. In order to republish an item, the following MUST occur:

1. Reparent the item sub-element from the **sx:unpublished** element to the **feed** element
2. If the **sx:unpublished** element contains no sub-elements, the **sx:unpublished** element SHOULD be removed.

### 5 Partial Feeds and Complete Feeds

Publishers will generally include, in a feed, only the most recent modifications, additions, and deletions within some reasonable time window. These feeds are referred to herein as *partial* feeds, whereas feeds containing the complete set of items are referred to as *complete* feeds.

In the feed sharing context new subscribers, or existing subscribers failing to subscribe within the published feed window, will need to initially copy a complete set of items from a publisher before being in a position to process incremental updates. As such, this specification provides for the ability for the latter feed to reference the complete feed. By placing the link to this feed in the feed descriptor, only the partial feed URL need to be distributed to potential subscribers.

The guidelines below explain how publishers and subscribers MAY optionally use **since** and **until** attributes for the **sx:sharing** element to ensure that all item updates are synchronized, even if the publisher periodically purges items from its feed. Note that these guidelines are only applicable when the subscriber maintains a local store for the feed’s items.

#### 5.1 Subscriber guidelines

###### 5.1.1 The following steps SHOULD be followed by the subscriber for initial feed consumption

* + - 1. Read the contents of the published feed
      2. See if a **sx:related** sub-element exists for the **sx:sharing** element, with a **type** attribute value of ‘complete’ – then repeat step 1 above with the feed URL specified by the **link** attribute
      3. Perform a merge operation (see Section 3.3)
      4. If the subscribed feed contains the **since** and **until** attributes for the **sx:sharing** element:
         1. The subscriber SHOULD cache the value of the **until** attribute for the **sx:sharing** element for subsequent feed consumption

###### 5.1.2 The following steps SHOULD be followed by the subscriber for subsequent feed consumption

* + - 1. Read the contents of the published feed
      2. If the feed contains the **since** and **until** attributes for the **sx:sharing** element and the subscriber cached the value of the **until** attribute from a previous read, then determine if the subscriber is out of sync with the feed:

1. If the value of the **since** attribute for the **sx:sharing** element is greater than the cached value of the **until** attribute:

If a **sx:related** sub-element exists for the **sx:sharing** element, with a **type** attribute value of ‘complete’ – if so then repeat step 1 above with the feed URL specified by the **link** attribute

* + 1. The subscriber is out of sync with feed, so all items that were not created, or most recently updated, by the subscribing endpoint should be discarded from the subscriber’s local store.
    2. Subsequent feed consumption is complete for the out of sync subscriber. Continue with the steps outlined for initial feed consumption in section 5.1.1

1. The subscriber SHOULD cache the value of the **until** attribute for the **sx:sharing** element for subsequent feed consumption
   1. Perform a merge operation for the feed items in the local store and the feed (see Section 3.3)

#### 5.2 Publisher guidelines

###### 5.2.1 The following steps SHOULD be followed by the publisher when publishing a feed for the first time (for either a complete or partial feed)

If items will ever be purged from the feed:

Set the value1 of the **since** attribute of the **sx:sharing** element to correspond to the incorporation of least recently updated item in the feed.

1. Set the value1 of the **until** attribute of the **sx:sharing** element to correspond to the incorporation of most recently updated item in the feed.
2. Optionally set the valueofthe **expires** attribute for the **sx:sharing** element to a date-time value before which subscribers SHOULD read the feed in order to avoid missing item updates due to items being purged from feed.

###### 5.2.2 The following steps MAY be followed by the publisher when subsequently publishing a feed

* + - 1. If the **expires** attribute for the **sx:sharing** element exists, update its value to a date-time before which subscribers SHOULD read the feed in order to avoid missing item updates.

###### 5.2.3 The following steps SHOULD be followed by the publisher when incorporating an item update into the feed

1. Increment the value of the **until** attribute for the **sx:sharing** element by an appropriate amount
2. Associate and store the new value of the **until** attribute withthe item
3. If the **expires** attribute for the **sx:sharing** element exists, optionally keep track of the date-time when incorporating the item’s changes. This value MAY be used in relation to the **expires** attribute for the **sx:sharing** element by the publisher when determining which items to remove from a feed.

###### 5.2.4 The following steps SHOULD be followed by the publisher when removing an item from a feed

* + - 1. If the stored value associated with the item from (step 2 in section 5.2.3) is greater than the value for the **since** attribute for the **sx:sharing** element, set the value for the **since** attribute for the **sx:sharing** element to the stored value

1**since** and **until** values

These values SHOULD be thought of as ever-increasing values that are represented in a manner that allows string collation by subscribers to yield the appropriate results. For example, if the publisher uses a date-time value to represent an incorporation timestamp for **since** and **until**, the value MUST be normalized for string comparison (e.g. “02-05-2006T09:10:33Z” instead of “2 May 2006 09:10:33”).

### 6 Licensing Information

Microsoft's copyrights in this specification are licensed under the Creative Commons Attribution-ShareAlike License (version 2.5). To view a copy of this license, please visit http://creativecommons.org/licenses/by-sa/2.5/. As to software implementations, Microsoft is not aware of any patent claims it owns or controls that would be necessarily infringed by a software implementation that conforms to the specification's extensions. If Microsoft later becomes aware of any such necessary patent claims, Microsoft also agrees to offer a royalty-free patent license on reasonable and non-discriminatory terms and conditions to any such patent claims for the purpose of publishing and consuming the extensions set out in the specification.