Frequently Asked Questions for Simple Sharing Extensions (SSE)

Frequently Asked Questions for Simple Sharing Extensions (SSE)

**General**

**What kinds of scenarios does SSE enable?**

Just as RSS and Atom enable the aggregation of information from a variety of data sources, SSE enables the synchronization of information across a variety of data sources. Data sources that implement SSE will be able to exchange and synchronize data with any other data source that also implements SSE.

From the user's perspective, this means that you will be able to share *your data* (such as calendar appointments, contact lists, and favorites) across all of your devices or services and with anyone else that you choose, regardless of infrastructure or organization.

SSE is particularly useful for scenarios in which there are multiple masters and/or asynchronous updates. For example, SSE could be used to share your work calendar with your spouse--either of you could enter new appointments, even if not currently connected. Similarly, SSE could be used to synchronize a set of calendar entries among a group of people, each working in a different company and using different infrastructure.

**What is the Simple Sharing Extensions specification?**

Simple Sharing Extensions (SSE) is a specification that extends existing formats such as RSS or Atom from unidirectional to bidirectional information flows. This version of the specification describes a “binding” for SSE to Atom and RSS; future versions of the spec are expected to include SSE bindings for other formats as well.

SSE defines the *minimum* extensions necessary to enable loosely cooperating applications or services to use formats such as RSS or Atom as the basis for *item sharing*--that is, the bidirectional, asynchronous synchronization of new and changed items among two or more cross-subscribed feeds.

For example, SSE could be used to share your work calendar with your spouse. If your calendar were published to an SSE feed, changes to your work calendar could be synchronized to your spouse's calendar, and vice versa. As a result, your spouse could see your work schedule and add new appointments, such as a parent-teacher meeting at the school, or a doctor's appointment.

SSE allows you to synchronize any set of independent items (for example, calendar entries, lists of contacts, list of favorites, blogrolls) using simple RSS or Atom semantics. If you can publish your data as a feed, the simple addition of SSE will allow you to synchronize your data to any other application or service that implements the SSE specification.

**When should I use SSE as opposed to some other synchronization technology?**

SSE is ideal for synchronizing data that can be published in an RSS or Atom feed. That includes collections of items such as calendar appointments, contacts, favorites, and news items. In addition, SSE is ideal for bidirectional, asynchronous synchronization, particularly for scenarios where multiple people can independently edit or create entries--where there is not a single “master” copy of the data and each end user has their own copy of the data. SSE was designed with simplicity in mind, and developers will find that implementing SSE synchronization is easy, especially for an application that already supports RSS or Atom.

**Are there any license restrictions to using SSE?**

Microsoft has released the Simple Sharing Extensions specification under the Creative Commons license. This license lets others remix, tweak, and build upon the specification even for commercial reasons, as long as they credit Microsoft, and license changed specifications under the same terms. Read [this](http://creativecommons.org/licenses/by-sa/2.5/) for more information about the Creative Commons license.

**What’s new in the 0.93 version of the SSE Spec?**

The major changes include:

* Addition of the Atom binding
* Updates to the synchronization algorithm for improved accuracy and efficiency
* Removal of the OPML binding
* Reorganization of the spec for improved readability

**Why did you add an Atom binding for SSE?**

In the time since we released the original SSE spec, Atom has seen wider adoption. SSE was always designed to be attached to multiple container formats. Given the widespread use of Atom, enabling Atom feeds to use SSE was a logical step.

**Why did you remove the OPML binding for SSE?**

The support for hierarchies in OPML made it challenging to support sync correctly, and still maintain the simplicity that is important for SSE. After considering the feedback on OPML from the original spec, we decided to remove the OPML binding from the first version. If there is interest in the community post-v1, we will consider adding OPML support to a future version of the SSE spec.

**What does it mean for the SSE specification to be at version 0.93?**

The SSE specification published here is a draft specification. We are seeking feedback on how this draft SSE extension to RSS and Atom can be improved. We expect to produce the 1.0 revision of the spec in the near future. To discuss the SSE spec or for questions, please send mail to the FEED-TECH list on <http://discussms.hosting.lsoft.com/scripts/wa-MSD.exe?INDEX>.

**Technology**

**How does SSE work?**

SSE defines new XML elements that add synchronization information to items in an RSS or Atom feed. These new elements enable SSE applications conforming to the specification to implement bidirectional feeds. In other words, two endpoints can mutually publish and subscribe to each other's feeds. When changes are made in one endpoint, they are propagated to the other and vice versa.

The new XML elements described in SSE 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). SSE also allows preservation of conflicting data to preserve data state for later resolution by end users.

**What kinds of topologies are supported by SSE?**

Any topology of feed interconnection is supported, from a hub-and-spoke to a star, to loops, to an arbitrary mesh. Item update propagation time or reliability might vary depending upon the topology, or the symmetry of feed update periods between cross-subscribed feeds, but the ultimate consistency of all feeds is assured. No "master feed" is required; there is no "true state" retained in a centralized manner.

**How many endpoints can participate in SSE synchronization?**

Any number of applications and devices can participate, representing any number of users. A single collection of items can be shared by two people or two thousand, each being a potential reader or writer.

**Can endpoints make changes asynchronously?**

As in standard RSS and Atom, feeds can be updated asynchronously. There is no requirement that changes be made online in order to assure consistency.

**Does this mean that I can sync an RSS feed to an Atom feed?**

While this is possible, because of the differences between RSS and Atom, we recommend that applications choose either RSS or Atom feeds to use for sync, but not both. For instance, a feed publisher could provide an SSE enabled RSS feed for two-way sync, and still provide a non-SSE Atom feed for one-way publishing, or vice versa.

**What protocols does SSE support?**

The focus is on the data format; HTTP/HTTPS is not required as a transport protocol. As long as a publisher's feed files and their dependencies (such as enclosures) can be transported to subscriber (for example, through a store-and-forward relay), any protocol will work.

**Is SSE secure?**

SSE defines a set of XML elements to express synchronization information in an RSS or Atom feed. SSE does not define a protocol or a means for communication. It is possible to transmit SSE information over any protocol, including secure ones such as SSL, WS-Secure, etc.

**Does SSE scale?**

Yes. Simple Sharing Extensions are designed to extend RSS and Atom feeds and as such scale to the same degree.

**Implementation**

**Where can I find sample code?**

Sample code is available today for download from MSDN as part of the [SSE Tutorials](http://www.microsoft.com/downloads/details.aspx?familyid=9bc8885e-76ba-4f10-8ac7-6f5845adcfd9&displaylang=en). We expect to publish additional sample code around the time when we finalize the 1.0 version of the specification.

**How often should a subscribing SSE application or service poll for Feed changes?**

As in any polling system, the average latency that a user will experience to get a notification will be one-half of the polling interval. For some of the examples here, such as calendar synchronization, a polling interval of minutes is appropriate. For other use-cases, such as synchronizing a contact list or (perhaps) favorites list, a polling interval of hours is sufficient.

The publishers can use the **window** attribute in the **sx:sharing** element to express the size of the window of change history kept by the publisher. Subscribers should poll at least this frequently.

**How much item history should I keep?**

Publishers should keep as much item history as is practical. For most of the scenarios in which SSE is used (calendar synchronization, for example) we recommend that the publisher keep all history. The latest revision of the 0.93 spec now allows for “sparse purging” of item change history.

**When should I purge deletion tombstones?**

Deletion tombstones can be purged if they are older than the window of change history kept by the publisher.

**Should the sx:sync id attribute match the atom:id element?**

When you use SSE with Atom feeds, every Atom entry has a unique id element, and the sx:sync element within the entry element has a unique id attribute. The reason that we require the unique id attribute in sx:sync is because SSE will be used in contexts other than Atom (such as RSS, but other formats as well) where it is not required or common to have a unique id.

It is reasonable (but not required) for both ids to be the same. SSE consumers should not assume that the ids are the same, and must be able to handle feeds where the two ids are different.

**What does it mean that an endpoint id must conform to the Namespace Specific String syntax from RFC 2141?**

Endpoint identifiers are used in two attributes: the “id” attribute in the sx:sync element, and the “by” attribute in the sx:history element. The requirement in section 1.2, item 10 states that “All endpoint identifiers MUST conform to the syntax for Namespace Specific Strings (the NSS portion of a URN) in RFC 2141.” In practice, what this means is that the endpoint id could be a simple string like “ABC123”, or it could be a URN such as a UUID. Both examples meet the NSS syntax requirement. It is up to the application developer to decide which type of endpoint id makes more sense for their application. For simple applications, it might be possible to ensure that the “id” and “by” attributes are unique without having to use UUIDs. For more complex applications, the application developer might decide that a UUID or URL is more appropriate.

**What’s the difference between partial and complete feeds?**

The SSE spec differentiates between partial and complete feeds in order to improve communication efficiency. Consider the typical sync conversation between two endpoints: after the initial exchange of feed data between the two endpoints, each endpoint has a complete feed representing the full data set. Successive sync operations between the two endpoints would be more efficient if each endpoint had a way to only send the changed data to the other side. This is the role of partial feeds.

When an SSE consumer reads a partial feed, it views that feed as a set of changes to incorporate into the complete data set. Likewise, an SSE publisher produces a partial feed to indicate that the feed only contains a subset of the data. It is up to the application developer to determine exactly what to include in a partial feed. The way to indicate that a feed is partial is by including the sx:related element inside the sx:sharing element.

We expect that another common way to improve efficiency after the first full sync is to have some filtering mechanism in the protocol used to retrieve or publish a feed. For example, HTTP has a conditional GET mechanism which allows a Web server to return data to a caller only if the page has changed, and then only return the new data. We would encourage anyone publishing SSE feeds over HTTP to use this mechanism. In this case, the SSE feed produced will be marked as “complete” because it represents the “complete” feed requested by the client.