**UpdateXmlMessages**

**XML Messages for Updating a Solr Index**

Solr accepts POSTed XML messages that Add/Replace, Commit, Delete, and Delete by query, using the url `/update` (there is also a CSV interface). Here is the XML syntax that Solr expects to see:

- **XML Messages for Updating a Solr Index**
  - The Update Schema
    - add/replace documents
      - Optional attributes for "add"
      - Optional attributes on "doc"
      - Optional attributes for "field"
      - Examples of adding docs with various optional attributes
    - "commit" and "optimize"
      - Optional attributes for "commit" and "optimize"
      - Optional attributes for "commit"
      - Optional attributes for "optimize"
    - Examples of "commit" and "optimize" with optional attributes
    - Passing commit and commitWithin parameters as part of the URL
    - "delete" documents by ID and by Query
      - Optional attributes for "delete"
    - "rollback"
    - "prepareCommit"
    - Updating a Data Record via curl
    - Updating via GET
    - Add and delete in a single batch

**The Update Schema**

(Not to be confused with schema.xml.)

**add/replace documents**

Simple Example:

```xml
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="office">Bridgewater</field>
    <field name="skills">Perl</field>
    <field name="skills">Java</field>
  </doc>
</add>
```

Git contains many complex examples of %3Cadd%3E document messages.

Note: multiple documents may be specified in a single <add> command.

**Optional attributes for "add"**

- overwrite = "true" | "false" — default is "true", meaning newer documents will replace previously added documents with the same uniqueKey.
- commitWithin = "(milliseconds)" if the "commitWithin" attribute is present, the document will be added within that time. ^ Solr1.4. See CommitWithin
- (removed in Solr4.0 - use "overwrite") allowDups = "true" | "false" — default is "false"
- (removed in Solr4.0 - use "overwrite") overwritePending = "true" | "false" — default is negation of allowDups
- (removed in Solr4.0 - use "overwrite") overwriteCommitted = "true" | "false" — default is negation of allowDups

**Optional attributes on "doc"**

- boost = <float> — default is 1.0
  - This is a convenience mechanism equivalent to specifying a boost attribute on each of the individual fields that support norms (see below)

**Optional attributes for "field"**

- update = "add" | "set" | "inc" — for atomic updating and adding of fields ^ Solr4.0
- boost = <float> — default is 1.0 (See SolrRelevancyFAQ)
Examples of adding docs with various optional attributes

**Example of “add” with optional boost attribute:**

```
<add>
  <doc boost="2.5">
    <field name="employeeId">05991</field>
    <field name="office" boost="2.0">Bridgewater</field>
  </doc>
</add>
```

**Example of “add” with optional update attribute:**

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="office" update="set">Walla Walla</field>
    <field name="skills" update="add">Python</field>
  </doc>
</add>
```

**Example of “add” with optional update attribute to set multiple values on a multi-valued field:**

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="skills" update="set">Python</field>
    <field name="skills" update="set">Java</field>
    <field name="skills" update="set">Jython</field>
  </doc>
</add>
```

**Example of “add” with optional update attribute to set a field to null (i.e. delete a field):**

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="skills" update="set" null="true" />
  </doc>
</add>
```

"commit" and "optimize"

A commit operation makes index changes visible to new search requests. A **hard commit** also calls fsync on the index files to ensure they have been flushed to stable storage and no data loss will result from a power failure.

A **soft commit** is much faster since it only makes index changes visible and does not fsync index files or write a new index descriptor. If the JVM crashes or there is a loss of power, changes that occurred after the last hard commit will be lost. Search collections that have near-real-time requirements (that want index changes to be quickly visible to searches) will want to soft commit often but hard commit less frequently.

An **optimize** is like a hard commit except that it forces all of the index segments to be merged into a single segment first. Depending on the use cases, this operation should be performed infrequently (like nightly), if at all, since it is very expensive and involves reading and re-writing the entire index. Segments are normally merged over time anyway (as determined by the merge policy), and optimize just forces these merges to occur immediately.

Example:

```
<commit/>
<optimize/>
```

Optional attributes for "commit" and "optimize"
waitFlush = "true" | "false" — default is true — block until index changes are flushed to disk. At least in Solr 1.4 and later (perhaps earlier as well), this command has no affect. Solr 1.4 At least in Solr 1.4 and later
waitSearcher = "true" | "false" — default is true — block until a new searcher is opened and registered as the main query searcher, making the changes visible.
softCommit = "true" | "false" — default is false — perform a soft commit - this will refresh the 'view' of the index in a more performant manner, but without "on-disk" guarantees.

Optional attributes for "commit"

expungeDeletes = "true" | "false" — default is false — merge segments with deletes away. Solr 1.4

Optional attributes for "optimize"

maxSegments = N — default is '1' — optimizes down to at most this number of segments Solr 1.3

Example of "commit" and "optimize" with optional attributes

```xml
<commit waitSearcher="false")
<commit waitSearcher="false" expungeDeletes="true")
<optimize waitSearcher="false")
```

Passing commit and commitWithin parameters as part of the URL

Update handlers can also get commit related parameters as part of the update URL. This example adds a small test document and causes an explicit commit to happen immediately after:

```
```

This example will cause the index to be optimized down to at most 10 segments, but won't wait around until it's done (waitFlush=false):

```
```

"delete" documents by ID and by Query

Delete by id deletes the document with the specified ID. (ID here means the value of the uniqueKey field declared in the schema (in these examples, employeeld).

Delete by query deletes all the documents that match the specified query.

Example:

```xml
<delete><id>05991</id></delete>
<delete><query>office:Bridgewater</query></delete>
```

Note: The "delete by query" uses the Lucene query parser by default, so if you're trying to understand the results of delete by query, you might submit a URL like this:

```
?q=stuff nonsense&debugQuery=on
```

You should see something in the debug output like the following, along with the parsed query. Examining the parsed query may shed some light on any surprising results.

```xml
<str name="QParser">LuceneQParser</str>
```
In Solr 1.2, delete query is *much* less efficient than delete by id, because Solr has to do much of the commit logic each time it receives a delete by query request. In Solr 1.3, however, most of the overhead will have been removed.

⚠️ Solr 1.4 Both delete by id and delete by query can be specified at the same time.

Example:

```xml
<delete>
  <id>05991</id><id>06000</id>
  <query>office:Bridgewater</query>
  <query>office:Osaka</query>
</delete>
```

Optional attributes for "delete"

- (deprecated) fromPending = "true" | "false" — default is "true"
- (deprecated) fromCommitted = "true" | "false" — default is "true"

"rollback"

⚠️ Solr 1.4 Expert:

Example:

```xml
<rollback/>
```

The rollback command rollbacks all add/deletes made to the index since the last commit. It neither calls any event listeners nor creates a new searcher. This is an expert-level API that should only be used if the application is taking complete responsibility for update concurrency, replication, and sharding.

"prepareCommit"

⚠️ Solr 4.0 Expert:

The prepareCommit command is an expert-level API that calls Lucene's IndexWriter.prepareCommit().

Example:

```bash
curl 'http://localhost:8983/solr/update?prepareCommit=true'
```

Updating a Data Record via curl

You can use curl to send any of the above commands. For example:

```bash
  <doc boost="2.5">
    <field name="employeeId">05991</field>
    <field name="office" boost="2.0">Bridgewater</field>
  </doc>
</add>'
```

```bash
```

Until a commit has been issued, you will not see any of the data in searches either on the master or the slave. After a commit has been issued, you will see the results on the master, then after a snapshot has been pulled by the slave, you will see it there also.

Updating via GET

Short update requests can also be sent using a GET request (needs to be url-encoded) like:

```bash
```
Add and delete in a single batch

Mixing add and delete elements in a single batch will throw an Illegal to have multiple roots (start tag in epilog?) exception (SOLR-2277):

```
curl http://127.0.0.1:8983/solr/update/?commit=true -H "Content-Type: text/xml" --data-binary '<add><doc><field name="id">17</field></doc></add><delete><id>1234</id></delete>';  
```

Instead, the add and delete elements must be enclosed in within an update element:

```
curl http://127.0.0.1:8983/solr/update/?commit=true -H "Content-Type: text/xml" --data-binary '<update><add><doc><field name="id">17</field></doc></add><delete><id>1234</id></delete></update>';  
```