LivCos

# Cosmos Manipulation

## Table of Contents

# 1 Update Markup

## 1.1 Hierarchical Entities

These use cases allow to manipulate hierarchical structured entities within the cosmos.

The action commands select the context node(s) to work with (insert at, insert in, update, delete,...). They apply on all the nodes selected or non for empty sets.

Examples for the following use cases base on this XML data:

```
<addresses>
   <address id="1">
      <name>
         <first>John</first>
         <last>Smith</last>
      </name>
      <city>Houston</city>
      <country>United States</country>
      <phone type="home">333-300-0300</phone>
      <phone type="work">333-500-9080</phone>
      <note>This is a note</note>
      <!-- Some comment -->
   </address>
</addresses>
```

Most of the following examples are copies from published XUpdate Use Cases.

### 1.1.1 Create Node

Let's create and insert an element before,

```
<u:insert position="before" select="/addresses/address[@id = 1]/name/last">
   <middle>Lennox</middle>
</u:insert>
```

...after

```
<u:insert position="after" select="/addresses/address[@id = 1]/phone[@type='home']">
   <phone type="cell">490-494-4904</phone>
</u:insert>
```

...or as a child of the target node(s). The default attribute 'position="into"' appends the result at the end of the child node collection.

```
<u:insert select="/addresses/address[@id = 1]">
   <zip>90200</zip>
</u:insert>
```

Some like to add some decoration markup.

```
<u:insert select="/addresses/address[@id = 1]">
   <u:element name="zip">90200</u:element>
</u:insert>
```

To create an attribute, a processing instruction or a comment special markup is needed.

```
<u:insert select="/addresses/address[@id = 1]/phone[@type='home']">
   <u:attribute name="extension">223</u:attribute>
</u:insert>
<u:insert position="before" select="/addresses">
   <u:processing-instruction name="cocoon-process">type="xsp"</u:processing-instruction>
</u:insert>
<u:insert position="before" select="/addresses/address[@id = 1]/name">
   <u:comment>Another comment about the name</u:comment>
</u:insert>
```

While text or CDATA nodes can directly be created and inserted

```
<u:insert select="note">written in</u:insert>
<u:insert select="note"><![CDATA[ <html>]]></u:insert>
```

...or also with addition markup decoration.

```
<u:insert select="/addresses/address[@id = 1]/country">
   <u:text>of America</u:text>
</u:insert>
<u:insert select="/addresses/address[@id = 1]/note">
   <u:cdata><![CDATA[ with <tags and="attributes">]]></u:cdata>
</u:insert>
```

For elements and attributes we can also specifiy a namespace.

```
<u:insert select="/addresses/address[@id = 1]">
   <state>Texas</state>
</u:insert>
<u:insert select="/addresses/address[@id = 1]/note">
   <u:attribute name="importance" namespace="http://someDomain.org/someNamespace">high</u:attribute>
</u:insert>
```

### 1.1.2 Update Node

Updating a node actually replaces its content.

```
<u:update select="/addresses/address[@id = 1]/name/first">Johnathan</u:update>
<u:update select="/addresses/address[@id = 1]/phone[.='333-300-0300']/@type">cell</u:update>
<u:update select="/addresses/address[@id = 1]/comment()">A changed comment</u:update>
<u:update select="/addresses/address[@id = 1]/note/text()">Changed...</u:update>
```

An update on a name() path initiates a rename.

```
<u:update select="name()">new-element-name</u:update>
```

### 1.1.3 Replace Node

Similar to update, but this action replaces the whole node, instead of only its content.

```
<u:replace select="/addresses/address[@id = 1]/name/first/text()">Johnathan</u:replace>
<u:replace select="/addresses/address[@id = 1]/phone[.='333-300-0300']/@type">
   <u:attribute name="type">cell</u:attribute>
</u:replace>
<u:replace select="/addresses/address[@id = 1]/comment()">
   <u:comment>A changed comment</u:comment>
</u:replace>
<u:replace select="/addresses/address[@id = 1]/note/text()">Changed...</u:replace>
```

### 1.1.4 Delete Node

Deleting the selected nodes along with their content.

```
<u:delete select="/addresses/address[@id = 1]/phone[@type = 'home']"/>
<u:delete select="/addresses/address[@id = 1]/phone/@type"/>
<u:delete select="/addresses/address[@id = 1]/country/text()"/>
<u:delete select="/addresses/address[@id = 1]/comment()"/>
<u:delete select="/addresses/address[@id = 1]/name"/>
```

### 1.1.5 Create Fragment

We can also create an XML fragment with one command.

```
<u:insert select="/addresses">
   <address id="2">
      <name>
         <first>Susan</first>
         <last>Long</last>
      </name>
      <city>Tucson</city>
      <state>Arizona</state>
      <country>United States</country>
      <phone type="home">430-304-3040</phone>
   </address>
</u:insert>
```

### 1.1.6 Update Fragment

When we update a node with a fragment, like with a simple node update the selected node's content will be replaced.

```
<u:update select="/addresses/address[@id=1]/name">
   <first>Jonny</first>
   <last>Smitt</last>
</u:update>
```

### 1.1.7 Copy Node

To copy a node we simply insert a copy at the selected target.

```
<u:insert select="/addresses/address[@id=1]">
   <u:copy-of select="phone[@type='work']"/>
</u:insert>
```

Relative select-XPaths start in the context of the current select node. In this example the "phone" child element from the address serves for the copy source.

With this example we clone the "phone" element right after the old one.

```
<u:insert position="after" select="/addresses/address[@id=1]/phone[@type='work']">
   <u:copy-of select="."/>
</u:insert>
```

The copy-reference can also be used within fragments and with update commands.

### 1.1.8 Move Node

To move a node we insert itself at the selected target. It will be removed at its old location.

```
<u:insert position="after" select="/addresses/address[@id=1]/name/last">
   <u:move-from select="../first"/>
```

```
</u:insert>
```

Relative select-XPaths start in the context of the current select node. In this example the "first" child element from the "name" parent element will be moved after the selected "last" child element.

In this example the selected node (first name) switches place with its predecessor.

```
<u:insert position="after" select="/addresses/address[@id=1]/name/first">
   <u:move-from select="preceding-sibling::*[1]"/>
</u:insert>
```

The node-reference can also be used within fragments and with update commands.

```
<u:insert select="/addresses/address[@id=1]">
   <birth-name>
      <u:move-from select="name/first"/>
      <last>Miller</last>
   </birth-name>
</u:insert>
```

Be careful not to move the selected node itself!

```
<u:insert position="after" select="/addresses/address[@id=1]/name/last">
   <u:move-from select="../last"/>
</u:insert>
```

This update action results in an error, since the reference node to insert at has been moved by the "move-from" content.

### 1.1.9 Merge Node

-- not yet implemented --

In some situations we don't exactly know the nodes to modify. We don't know about the node in question exists already (update) or we need to create it first.

```
<u:merge select="/addresses/address[@id=1]">
   <name>
      <first>Jonny</first>
      <middle>B</middle>
      <last>Smith</last>
   </name>
</u:merge>
```

With this command the first name will be updated and the middle name will be created. The last name does not differ and will not be updated.

New nodes will simply be appended to the end of the list of child nodes, if the order is not defined with the command content. In the example the new middle name will be inserted before the matching last name.

Beside the default mode attribute value 'mode="join"' we can also cleanup things with a 'mode="intersect"' merge.

### 1.1.10 Rename Node

Rename named nodes (elements, attributes). Even though a content move into a new named node would also do the trick, the node would loose memory address and the "rename" action could not be handled individually (eg. create an identity proxy for the old name).

```
<u:rename select="/addresses/address[@id=1]/phone[@type = 'home']" name="home-phone"/>
<u:rename select="/addresses/address[@id=1]/phone/@type" name="place"/>
```

The expanded name of the node can also be specified by a qualified name and a namespace URI.

```
<u:rename select="/addresses/address[@id=1]/note" name="comment" namespace="http://someDomain.org/
      someDocNamespace"/>
```

Nodes other than elements and attributes cannot be renamed and will be ignored.

### 1.1.11 Declare Namespace

Even though namespaces can be handled through renames and inserts, we might still like to control any namespace declarations. Sometimes namespace declaration are not directly used in node naming (elements or attributes in this namespace). They could be necessary to interpret for example attribute values or alike.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:doc="http://someDomain.org/someDocNamespace">
   <xsl:template match="doc:*">...</xsl:template>
</xsl:stylesheet>
```

With a simple action we can declare such a namespace.

```
<u:decl-namespace select="/addresses" prefix="doc" uri="http://someDomain.org/someDocNamespace"/>
```

To remove a namespace declaration we simply declare it to nothing.

```
<u:decl-namespace select="/addresses" prefix="doc" uri=""/>
```

The NULL namespace URI "" can only be assigned to the default namespace prefix "".

```
<u:decl-namespace select="/addresses" prefix="" uri=""/>
```

The declaration of namespaces does not change any naming of the selected element or it's child nodes. Of course prefixes already in place will resolve according any new declarations.

## 1.2 Context

With contexts we can simplify the markup for multiple actions.

```
<u:context select="/addresses/address[@id = 1]">
   <u:update select="note">New Note</u:update>
   <u:delete select="comment()"/>
   <u:insert select=".">
      <comment>Some comment...</comment>
   </u:insert>
</u:context>
```

Any update action node also builds a context for it's content. See for example the "copy-of" or "move-from" content node.

### 1.2.1 Namespaces

In some situations we want to select certain nodes in a specific namespace. To specify a proper namespace context for XPath selects, we simply add namespace - prefix mappings to the update context.

```
<u:context select="xsl:template[4]">
   <u:namespace prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
   <u:_.../>
</u:context>
```

### 1.2.2 Create Path

When we don't know whether a path for a context completely exists, but we want missing nodes to be created, we can specify the "create-path" attribute.

```
<u:insert select="anElement/maybeNotExisting/anotherElement" create-path="true">
   <someNewElement/>
</u:insert>
```

All the elements of the hierarchy will be created, if not yet existing.

Even though the "select" attribute of a context can receive any valid XPath, for the "create-path" feature to work, only a simple element or attribute name path is allowed.

## 1.3 Object Nodes

Object nodes are elements in a special namespace (livcos.org/ns/cosmos) and can be addressed and manipulated like any other element.

Some extensions to the general update markup schema allow to handle object nodes more comfortably.

### 1.3.1 Context

To make it easier to select a certain object node as the working context, we can specify a "ref" attribute to expect an object ID instead of an XPath.

```
<u:context ref="/some.domain.com/someObject/Contacts">
   <u:context select="/addresses/address[@id = 1]">...</u:context>
</u:context>
```

In addition to the "ref" attribute we can specify the access scope to the object. By default the context addresses the object node itself ("meta" scope), but you can also access in the "content" scope. Since the result of the "view" scope is generated, it is not reasonable to manipulate.

Also the "copy-of" or "move-from" content node can address their source by a "ref" attribute.

```
<u:insert select=".">
   <ref>
      <u:copy-of ref="/some.domain.com/someObject/Contacts" scope="content" select="*[last()]/@id"/>
   </ref>
</u:insert>
```

Here we copy the generated ID from the last address in the Contacts object.

### 1.3.2 Create

We can insert an object node into a data access node, capable to create and store objects.

```
<u:insert select="da:folder">
   <u:namespace prefix="da" uri="livcos.org/ns/cosmos/data-access"/>
```

```
    <c:OtherContacts xmlns:c="livcos.org/ns/cosmos">
       <addresses>...</addresses>
    </c:OtherContacts>
 </u:insert>
```

The folder data access node creates a new object as a child of it's parent object and a corresponding file in it's file system path. The object is not a child of the data access node after the insert.

We can insert an object node into another, existing object node.

```
 <u:insert ref="/some.domain.com/someObject">
    <c:OtherContacts xmlns:c="livcos.org/ns/cosmos">
       <addresses>...</addresses>
    </c:OtherContacts>
 </u:insert>
```

The new object is not stored or managed by any data access node. Such objects can be used for temporary purposes only.

We can insert the content into a new object node.

```
 <u:insert ref="/some.domain.com/someNewObject/NewContacts" create-path="true">
    <addresses>
       <address id="address_1">...</address>
    </addresses>
 </u:insert>
```

Missing parent objects along the path will be created as well. The first data access node of the first existing object, found along the ancestor axes, handles the created object hierarchy.