# Fox Commands Reference

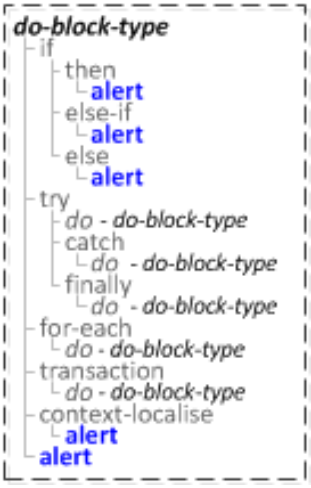
# FOX:Commands:COMMAND REFERENCE

- fm:XSLTransform
- fm:alert
- fm:apply
- fm:assign
- fm:attach
- fm:by
- fm:call-module
- fm:call
- fm:catch
- fm:compare
- fm:context-clear
- fm:context-localise
- fm:context-set
- fm:copy
- fm:do
- fm:else-if
- fm:else
- fm:eval
- fm:exit-module
- fm:finally
- fm:focus
- fm:for-each
- fm:generate
- fm:go-to-page
- fm:if
- fm:init
- fm:log
- fm:move
- fm:order
- fm:parse-spreadsheet
- fm:part
- fm:pragma
- fm:process-set
- fm:refresh-map-set
- fm:remove
- fm:rename
- fm:run-api
- fm:run-dml
- fm:run-query2
- fm:run-query
- fm:security-scope
- fm:send
- fm:set-cookie
- fm:state-pop
- fm:state-push

- fm:state-replace
- fm:state-strict-pop
- fm:state
- fm:then
- fm:throw
- fm:transaction
- fm:try
- fm:user-login
- fm:user-logout
- fm:validate
- fm:while

# FOX:Commands:alert

## Schema Location


## Description

Shows a browser alert to the user with the text specified in `message`.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| message | XPath String | The text that will be shown in the message box. | Yes |

## Examples

```
<fm:alert message="string"/>
```

## Notes

If the provided XPath String is a Complex XPath which returns null, a blank alert is displayed. Your code should ensure that the XPath always returns a string.

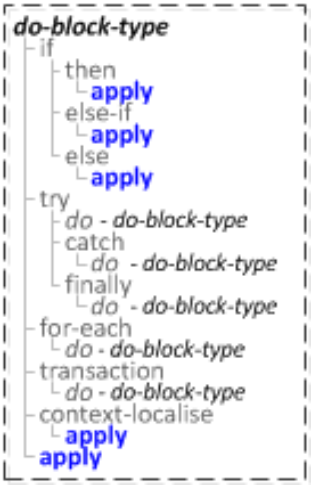
If you use any new lines then you have to convert them to '\n' otherwise it causes an unterminated string literal in javascript which breaks all the javascript in that script block, including selector mapsets and spatial maps.
eg.

```
alert('
');
```

If you use characters like ' and " they get double escaped resulting in &#39 and &#34 etc coming out in the alert.

# FOX:Commands:apply

## Schema Location


## Description

Applies field set changes.

## Syntax

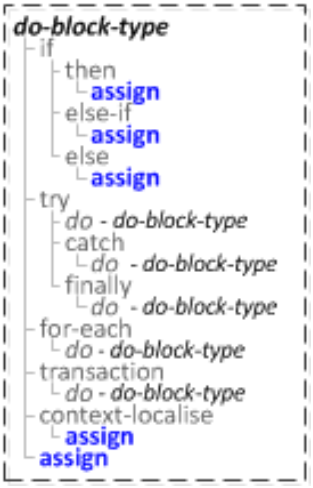
```
<fm:apply/>
```

## Examples

```
<fm:apply/>
```

# FOX:Commands:assign

## Schema Location

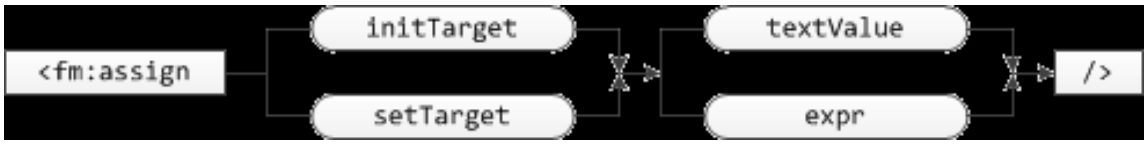




## Description

Assigns the String value provided in either the `expr` or `textValue` attributes to a Node identified by the `initTarget` or Node List identified by the `setTarget` attribute.

To refer to the Node being assigned to in the *expr* XPath, use the :{assignee} context.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| initTarget | Single Node Complex XPath | Will create this Node before an assignment if it does not exist. | When `setTarget` is **NOT** used. |
| setTarget | Complex XPath | Will assign to multiple Nodes if the expression returns a Node List. | When `initTarget` is **NOT** used. |
| expr | XPath String | Evaluates the XPath provided here as a string and assigns it to the node(s) identified in `initTarget` or `setTarget` | When `textValue` is **NOT** used. |
| textValue | String Literal | String value provided to assign to the node(s) identified in `initTarget` or `setTarget` | When `expr` is **NOT** used. |

## Examples

```
<fm:assign [initTarget="XPath" | setTarget="XPath"] [expr="XPath" | textValue="string"]/>
```

### Increment Value

```
<fm:assign setTarget="/PATH/TO/NODE" expr=":{assignee}+1"/>
```

This will increment the value of the NODE by 1.

### Assigning Sequential Numbers To A Node List

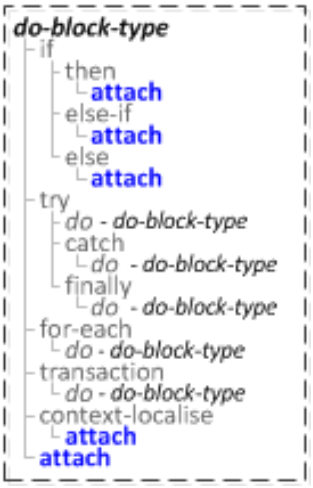
```
<fm:assign setTarget="/NODE_LIST/NODE/SEQ" expr="string(count(:{assignee}/../preceding-sibling::NODE)+1)"/>
```

This will assign the SEQ element of each "NODE" to be 1, 2, 3, etc. You need the string() part of the xpath otherwise you get 1.00, 2.00, 3.00, etc. For a reverse ordered sequence you can use following-sibling instead.

**Note:** You have to use the :{assignee} context in order to be in context of the correct part of the DOM.

# FOX:Commands:attach

## Schema Location


## Description

Changes the attach point for the current state.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| to | Single Node Complex XPath | The location in a DOM to attach to. | Yes |

## Examples

```
<fm:attach to="XPath"/>
```

Using following XML Structure:

```
<PARENT>
  <NAME>Bob</NAME>
  <AGE>34</AGE>
  <SEX>M</SEX>
  <CHILDREN>
    <CHILD>
      <NAME>Lucy</NAME>
      <AGE>7</AGE>
      <SEX>F</SEX>
    </CHILD>
    <CHILD>
      <NAME>Daniel</NAME>
      <AGE>12</AGE>
      <SEX>M</SEX>
    </CHILD>
    <CHILD>
      <NAME>Nicholas</NAME>
      <AGE>10</AGE>
      <SEX>M</SEX>
    </CHILD>
  </CHILDREN>
</PARENT>
```

```
<fm:attach to="/*/CHILDREN"/>
```

This would change the current attach point to be the CHILDREN list node. Such that if you then did a set-out command, with a match attribute of ".", FOX would set out the list of CHILD elements because . (current attach point) would point to the CHILDREN node.

# FOX:Commands:by

## Description

***fm:by*** is a child node of ***fm:order*** and provides the pivot parameters for sorting to occur.

## Syntax

```
<fm:by key="[ XPath Node ]" [logic="[ Sorting Logic ]"]/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| logic | String Literal | • `blank-number-alpha-ascend` - Nulls first, then in lexicographical order [1]<br>• `blank-number-alpha-descend` - Reverse order of blank-number-alpha-ascend.<br>The method to use in ordering the node list, when it is on the fm:order element it is used as the default for any `fm:by` keys without an explicit override.<br>**Default:** `"blank-number-alpha-ascend"` | No |
| key | Single Node Complex XPath | An xpath specifying a node whose value should be sorted by. | Yes |

## Examples

```
<fm:by key="FILE_ORDER" logic="blank-number-alpha-descent"/>
```

The above example will order the repeating element specified by ***fm:order*** by the contents of it's child node ***FILE_ORDER***. The sort order will be in descending order with nulls last.

## Notes

More information this commands use can be found under *fm:order

## Related

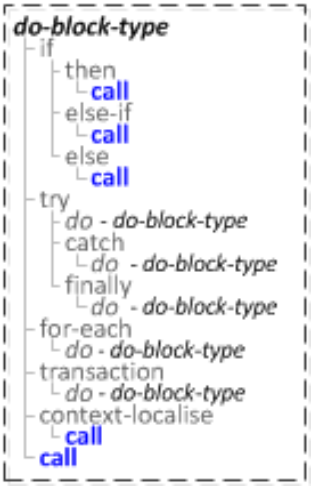
- fm:order

## References

[1] http://en.wikipedia.org/wiki/Lexicographical_order

# FOX:Commands:call

## Schema Location


## Description

Runs the action matching the given action name, if a state name is given it runs the matching action within that state.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| action | String Literal | The name of the action to call in the form "state-name/action-name" or just "action-name". | Yes |

## Examples

```
<fm:call action="String Literal"/>
```

### Calling a module-level action or another state action from within a state action

```
<fm:call action="action-doSomething"/>
```

This will run the action "action-doSomething" using action precedence ignoring the case where state-name is specified.

### Calling a state level action for a different state, or from a module level action

```
<fm:call action="state-otherState/action-doSomething"/>
```

This will run the action "action-doSomething" using action precedence including using the case where state-name is specified.

## Notes

### Action Precedence

1. **If state-name is specified:** State-level action from within the specified state using library precedence.
2. State-level action from within the current state.
3. Module-level action using library precedence.

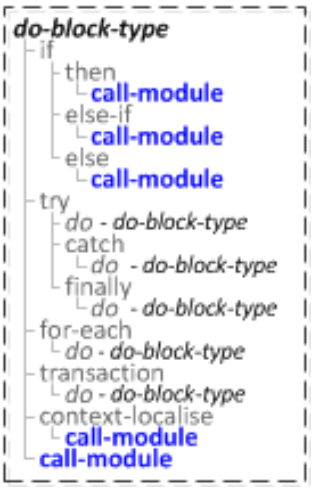
### Library Precedence

1. Base module
2. Libraries referenced by base module in order referenced within library-list
3. Libraries referenced by libraries in order referenced within library-list of base module, then of first-level library.

**Note:** It only uses the first instance of each state/action name found, it does not merge them in any way. Essentially works as if each time it hits a library include whilst processing any module it adds it to the end of a global queue and if it encounters the same state or same action names then it ignores any subsequent declarations.

# FOX:Commands:call-module

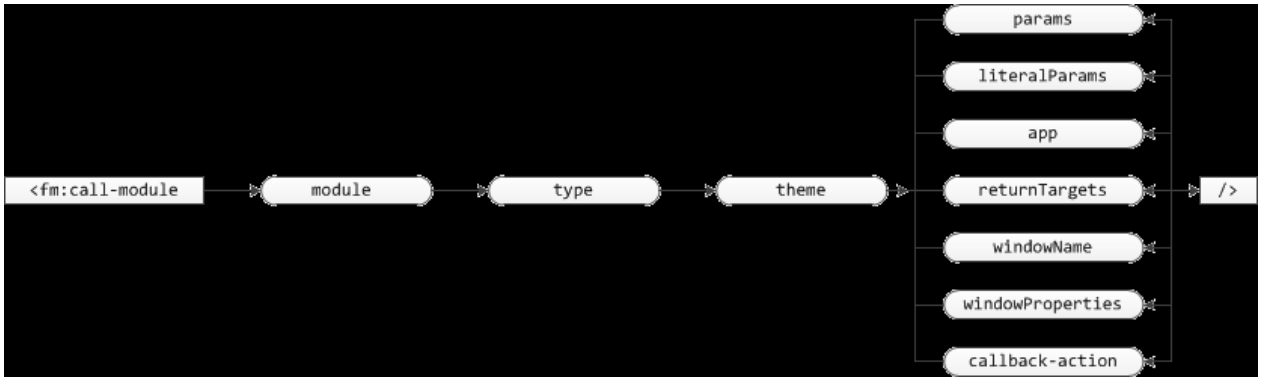
## Schema Location

## Description

Call another module, thus redirecting the user to another page.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| module | XPath String | Specifies the name of the module to be called as found under fm:name. | Yes |
| type | String Literal | • modal<br>• modeless<br>• modeless-orphan-same-session<br>• modeless-orphan-new-session<br>• modal-return-to-first-or-replace-all-cancel-callbacks<br>• modal-replace-all-cancel-callbacks<br>• modal-replace-this-cancel-callbacks<br>• modal-replace-this-caller-callbacks-now-then-cancel-callbacks<br>• modal-replace-this-preserve-caller-callbacks-for-exit<br>See Notes for descriptions of these options. | Yes |
| theme | XPath String | This specifies a valid entry-theme for the called module. | Yes |
| params | Complex XPath | XPath to Node(s) to be passed as parameters to the called module. These populate the called module's :{params} DOM. | No |
| literalParams | String Literal | Use to pass parameters to the called module in the form of a name=value comma-separated list, for example: "NAME=VALUE,NAME2=VALUE2,NAME3=VALUE3". | No |
| app | XPath String | This allows switching between app mnemonics, for example from your_app_server_one to your_app_server_two. Defaults to current app mnemonic. **Note:** This does not change the app mnemonic found under :{sys}/thread/app_mnem, as this pertains to the whole thread. To determine what app mnemonic an individual module was called under, examine :{sys}/module/app_mnem. | No |
| returnTargets | Complex XPath | The node list which is returned will have the contents of the :{return} DOM copied into each node upon the return to this module. | No |
| windowName | String Literal | This allows specifying the name for any child window popups, if a window with that name already exists it is reused. Defaults to a unique name. | No |
| windowProperties | String Literal | Only applicable to modeless module calls. A comma-separated list of properties for the child window that will display the module call dialog. These are javascript properties (see javascript reference manual on 'window' object) that specifiy the window geometry, etc. **Example:** windowProperties="width=200,height=600,resizable,status,toolbar,menubar" | No |

| | | | |
|---|---|---|---|
| callback-action | String Literal | This specifies the name of the action to run when the callstack returns to this module. | No |

## Examples

```
  <fm:call-module module="XPath String" type="string" theme="XPath String"
    [params="XPath Node List" literalParams="string" app="XPath String"
 returnTargets="XPath Node List"
    windowName="string" windowProperties="string"
callback-action="string"]/>
```

## Notes

### `type` Attribute

The two main types of module call are **modal** and **modeless**. Modal modules appear in the same browser window and share their module callstack with the calling module. Modeless modules appear in a separate browser window with an autonomous callstack.

The table below provides detailed descriptions for each type of module call.

| Call Type | Description |
|---|---|
| **modal** | The most common way to call a module. All callback actions are preserved and the called module is displayed in the same window as the calling module. Exiting the called module via `fm:exit-module` or `fm:state-pop` pops the callstack back to the calling module and runs the `callback-action` as defined on the `fm:call-module` command. It also populates the :{result} DOM and any nodes targeted in the `returnTargets` attribute. |
| **modal-replace-all-cancel-callbacks** | Replaces all the modules on the callstack with the called module and does not run any callbacks. |
| **modal-replace-this-cancel-callbacks** | Replaces the calling module with the called module. If the calling module was itself called with a callback-action, it is cancelled. |
| **modal-replace-this-caller-callbacks-now-then-cancel-callbacks** | Replaces the calling module with the called module. If the calling module was itself called with a callback-action, it is run now. It will not be run when the newly-called module exits. Note: any callstack transformations caused by running the callback-action are ignored. |
| **modal-replace-this-preserve-caller-callbacks-for-exit** | Replaces the calling module with the called module. If the calling module was itself called with a callback-action, this will be run when the newly-called module exits. The module running the callback action should be aware that its :{result} DOM may have been populated from different sources. |
| **modal-return-to-first-or-replace-all-cancel-callbacks** | Returns to the first occurrence of the desired module on the callstack. If the module being called is already at the top of the callstack, the command will do nothing. If the called module does not currently exist in the callstack then it replaces all the modules currently on the callstack. In any case, no callback actions are run. |

| | |
|---|---|
| **modeless** | ~~Opens the called module in a new browser window. The calling module remains available in the original window. The called module is on a new callstack, but the same underlying FOX Session is used by both. This means they share the same :{session} DOM, and their transaction cycles are processed sequentially (i.e. Module 1 will wait for any action processing from Module 2 to finish before it can do any work).~~<br>~~You can define a `callback-action` or `returnTargets` on a modeless module call, but it is not recommended. When the called module is exited (via `fm:exit-module` or `fm:state-pop`), it will attempt to run the callback-action on the calling module, or fill in any `returnTargets`, if either were defined. However, there is no guarantee that the base window will still be displaying the calling module. If it is not, the user will see an error message.~~<br>~~If the user has forcibly closed the browser window (and not run a FOX action to cause it to clause), no callback-action is run.~~<br>As of FOXr4.04.32, this call type is no longer supported. Where it is currently used in module definitions, it is substituted within FOX for **modeless-orphan-same-session**. |
| **modeless-orphan-same-session** | Opens the called module in a new browser window. The calling module remains available in the original window. The called module is on a new callstack, but the same underlying FOX Session is used by both. This means they share the same :{session} DOM, and their transaction cycles are processed sequentially (i.e. Module 1 will wait for any action processing from Module 2 to finish before it can do any work).<br>The called module has no awareness of its caller, so it is invalid to define a `callback-action` or `returnTargets` for this type of module call. However, both modules share the same FOX Session, so it is possible to use this for inter-module communication. |
| **modeless-orphan-new-session** | As above, although the called module is completely independent and is launched in a new session. This is allows the end user to process transactions concurrently without having to manually open a new browser window. |

For more details on how FOX perform session management, consult the Session Concepts page.

## Passing data between modules

You can pass parameters to the called module using `params` or `literalParams`.

Typically, when the called module is exited, the contents of its :{return} DOM are populated into the calling module's :{result} DOM. At this point, the `callback-action` is run if defined. It is therefore possible to process the results from calling a module with a callback-action.

For simpler requirements, the `returnTargets` attribute defines a list of nodes (typically 1) to which the contents of the :{result} DOM is copied automatically.

# FOX:Commands:catch

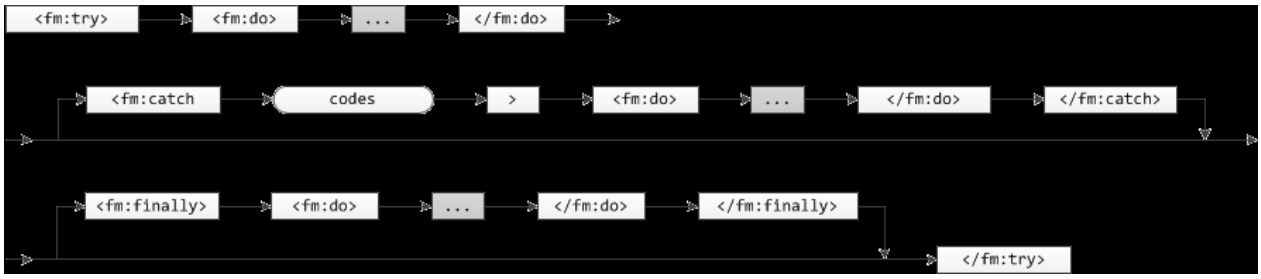
## Schema Location



## Description

Try/catch/finally, similar to Java try/catch/finally blocks.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| code | String Literal | A list of space/tab separated codes to catch, as defined by the code parameter of fm:throw. | Yes |

## Examples

```
<fm:try>
  <fm:do>
    ...
    [command-list]
    ...
  </fm:do>
  [<fm:catch codes="string">
    <fm:do>
      ...
      [command-list]
      ...
    </fm:do>
```

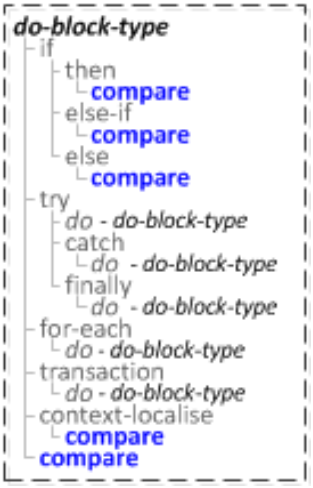
```
  </fm:catch> ...]
  [<fm:finally>
    <fm:do>
      ...
      [command-list]
      ...
    </fm:do>
  </fm:finally>]
</fm:try>
```

### Notes

See fm:throw for how to raise an error.

# FOX:Commands:compare

## Schema Location


## Description

Compare the data in context-one with the data in context-two and show the differences in context-out. The differences are identified by copying the XML from context-one and adding 'fox-history' xml nodes each node that differs between context-one and context-two.

The fox history take the following structure and appears after any data within the element.

```
<SOME_XML_ELEMENT>
  value-from-context-one
  <fox-history>
    <history>
      <label>hello</label>
      <value>value-from-context-two</value>
      <operation>update</operation>
    </history>
  </fox-history>
</SOME_XML_ELEMENT>
```

The operation node suggests whether this node was updated (`update`), inserted (`insert`) or deleted (`delete`).

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| context-one | Single Node Complex XPath | Context for the XML of the data to compare with context-two. When comparing historical records/edited data, this will be the newer/edited data. | Yes |
| context-two | Single Node Complex XPath | Context for the XML of the data to compare with context-one. When comparing historical records/edited data, this will be the older/original data. | Yes |
| context-out | Single Node Complex XPath | Context for the location of the results of the comparison. | Yes |
| version-two | XPath String | Result of this XPath when evaluated is used as the `label` in the fox-history XML. | Yes |
| schema-module | XPath String | **Default:** Current module. | No |

## Examples

```
<fm:compare context-one="XPath" context-two="XPath" context-out="XPath" version-two="XPath" [schema-module="XPath"]/>
```

Using the following XML structure:

```
<theme>
  <REVIEW_LIST>
    <REVIEW>
      <HR_REVIEW_GROUP_ID>300</HR_REVIEW_GROUP_ID>
      <TEAM_TITLE>Energy Markets &amp;amp; Infrastructure</TEAM_TITLE>
      <STATUS>ISSUED</STATUS>
      <ISSUED_DATE>2010-10-05T13:43:55</ISSUED_DATE>
    </REVIEW>
    <REVIEW>
      <HR_REVIEW_GROUP_ID>300</HR_REVIEW_GROUP_ID>
      <TEAM_TITLE>Energy Markets &amp;amp; Infrastructure</TEAM_TITLE>
      <STATUS>DRAFT</STATUS>
      <CREATED_BY>1000</CREATED_BY>
    </REVIEW>
  </REVIEW_LIST>
</theme>
```

And the following FOX commands:

```
<fm:context-localise name="one" xpath=":{theme}/REVIEW_LIST/REVIEW[1]" operation="reset-references">
  <fm:context-localise name="two" xpath=":{theme}/REVIEW_LIST/REVIEW[2]" operation="reset-references">
    <fm:context-localise name="three" xpath=":{theme}/RESULTS" operation="reset-references">
      <fm:compare context-one=":{one}" context-two=":{two}" context-out=":{three}" version-two=":{two}/HR_REVIEW_GROUP_ID"/>
    </fm:context-localise>
```

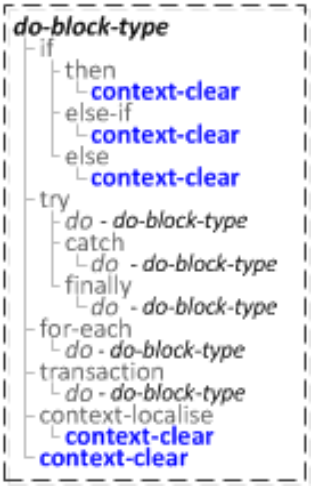
```
    </fm:context-localise>
  </fm:context-localise>
```

Produces the following XML (in addition to the ogirinal XML in the DOM:

```
<theme>
  <REVIEW_LIST>
    <REVIEW>
      <HR_REVIEW_GROUP_ID>300</HR_REVIEW_GROUP_ID>
      <TEAM_TITLE></TEAM_TITLE>
      <STATUS>
      ISSUED
        <fox-history>
          <history>
            <label>300</label>
            <value>SUBMITTED</value>
            <operation>update</operation>
          </history>
        </fox-history>
      </STATUS>
      <ISSUED_DATE>
      2010-10-05T13:43:55
        <fox-history>
          <history>
            <label>300</label>
            <operation>insert</operation>
          </history>
        </fox-history>
      </STATUS>
      <CREATED_BY>
      1000
        <fox-history>
          <history>
            <label>300</label>
            <operation>delete</operation>
          </history>
        </fox-history>
      </CREATED_BY>
    </REVIEW>
  </REVIEW_LIST>
</theme>
```

# FOX:Commands:context-clear

## Schema Location


## Description

Clears a given context so that it can no longer be referenced within future XPaths.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| scope | String Literal | • localised - Used only when inside fm:context-localise and fm:for-each blocks.<br>• scope - Clears the context for the current state, used everywhere else. | Yes |
| name | String Literal | The name of the context to clear, eg if you use the name "test" it clears the ":{test}" context. Can also be used to clear the in-built contexts other than :{root} | Yes |

## Examples

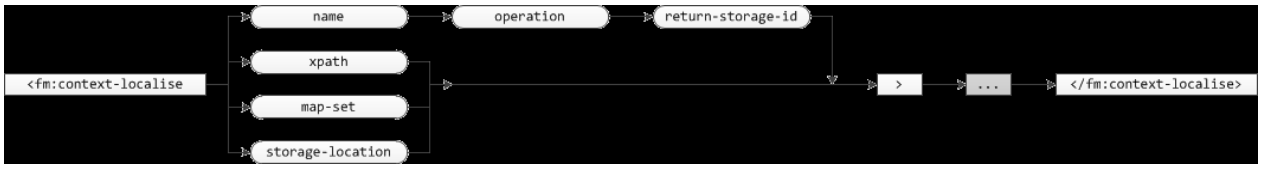
```
<fm:context-clear scope="string" name="string"/>
```

# FOX:Commands:context-localise

## Schema Location



## Syntax




## Attribute Summary

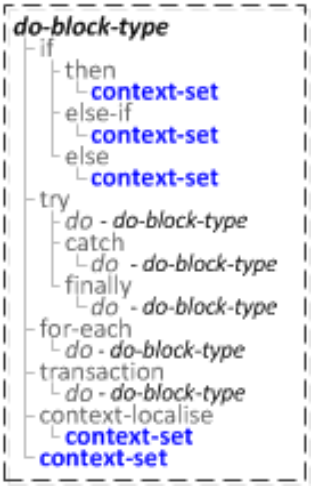
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String Literal | **Default:** "local" | No |
| operation | String Literal | • reset-references | Only when `name` is used. |
| return-storage-id | Single Node Complex XPath | | Only when `name` is used. |
| xpath | Single Node Complex XPath | **Default:** "." | No |
| map-set | String Literal | | No |
| storage-location | String Literal | | No |

## Examples

```
<fm:context-localise [name="string" operation="string" return-storage-id="XPath"] [xpath="XPath" | map-set="string" | storage-location="string"]>

  ...

  [command-list]

  ...

</fm:context-localise>
```

# FOX:Commands:context-set

## Schema Location


## Description

Sets a context that can be referenced within future xpaths.

## Syntax


## Attribute Summary

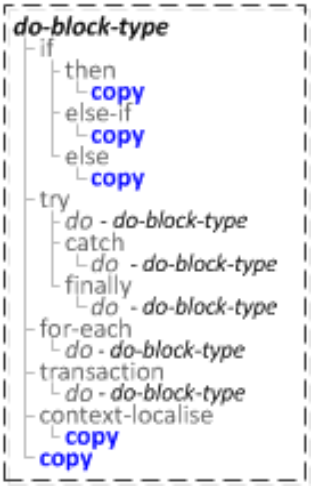
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| scope | String Literal | • localised - Used only when inside fm:context-localise and fm:for-each blocks.<br>• state - Sets the context for the current state only, used everywhere else. | Yes |
| name | String Literal | The name to use to reference the context, eg if you use the name "test" you can refer to it with ":{test}". Can also be used to override the default contexts such as :{action}. | Yes |
| xpath | Single Node Complex XPath | The node for the context to reference. | Yes |

## Examples

```
<fm:context-set scope="string" name="string" xpath="XPath"/>
```

# FOX:Commands:copy

## Schema Location


## Description

Duplicates all targeted source nodes from `from` nodes to `to` nodes

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| from | Complex XPath | Location to copy nodes from.<br>**Defaults to:** "." | No |
| to | Complex XPath | Location to copy nodes to.<br>**Defaults to:** "." | No |

## Examples

```
<fm:copy [from="XPath"] [to="XPath]"]/>
```

Using following XML Structure:

```
<PARENT>
  <NAME>Bob</NAME>
  <AGE>34</AGE>
  <SEX>M</SEX>
  <CHILDREN>
    <CHILD>
      <NAME>Lucy</NAME>
```

```
        <AGE>7</AGE>
        <SEX>F</SEX>
      </CHILD>
      <CHILD>
        <NAME>Daniel</NAME>
        <AGE>12</AGE>
        <SEX>M</SEX>
      </CHILD>
      <CHILD>
        <NAME>Nicholas</NAME>
        <AGE>10</AGE>
        <SEX>M</SEX>
      </CHILD>
    </CHILDREN>
  </PARENT>
```

```
<fm:copy from="/*/CHILDREN" to=":{theme}"/>
```

This would put the entire CHILDREN list from the root DOM in the theme DOM.

# FOX:Commands:do

## Description

***fm:do*** is a container for FOX Commands (a full list can be found in the FOX:Commands section). All elements inside an ***fm:do*** are executed sequentially and behave much like the contents of a method in a 3GL and effectively generate a call-stack of operations to complete. Sub Commands (such as ***fm:call*** action) will be placed on top of the call-stack, again in a like fashion to 3GL's.

## Syntax

```
<fm:do>
  [FOX: Commands]
</fm:do>
```

## Examples

```
<fm:do>
  <fm:call action="action-dummy"/>
  <fm:alert message="Action has finished"/>
</fm:do>
```

## Notes

There's a multitude of locations in the FOX Module markup that this commands can be called from, you'll find annotations in the appropriate section if said markup can utilize ***fm:do***.

## Related

- fm:entry-theme
- fm:action
- Do Command Reference

# FOX:Commands:else

## Description

Commands nested within <fm:else> will be executed if the parent <fm:if> test attribute is evaluated to false.

## Attribute Summary

*None*

## Examples

```
<fm:if test="something false">
  <fm:then>
    not executed
  </fm:then>
  <fm:else>
    code to execute
  </fm:else>
</fm:if>
```

## Related

- <fm:if>
- <fm:then>
- <fm:else-if>

# FOX:Commands:else-if

## Description

<fm:else-if> is contained in a <fm:if> command, and is eveluated if the parent <fm:if> test attribute evaluates to false.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| test | XPath Boolean | The XPath expression to test. | Yes |

## Examples

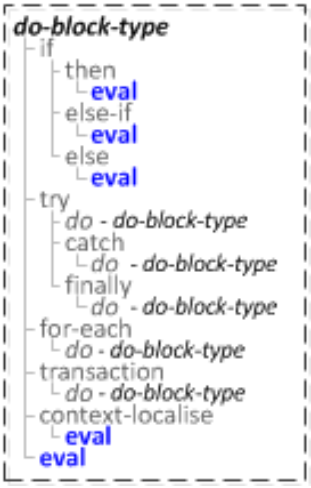
```
<fm:if test="evaluates to false">
  <fm:then>
    not executed
  </fm:then>
  <fm:else-if test="true">
     code to execute
  </fm:else-if>
  <fm:else>
    not executed
  </fm:else>
</fm:if>
```

## Related

- <fm:if>
- <fm:then>
- <fm:else>

# FOX:Commands:eval

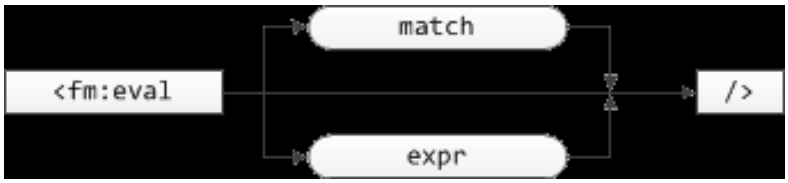
## Schema Location


## Description

Dynamically evaluates/executes the FOX command(s) provided by the *expr* or *match* attributes.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | Complex XPath | An xpath to a list of FOX command nodes to evaluate/execute. | No |
| expr | XPath String | A string of FOX commands to evaluate/execute. | No |

## Examples

```
<fm:eval [match="XPath" | expr="XPath"]/>
```

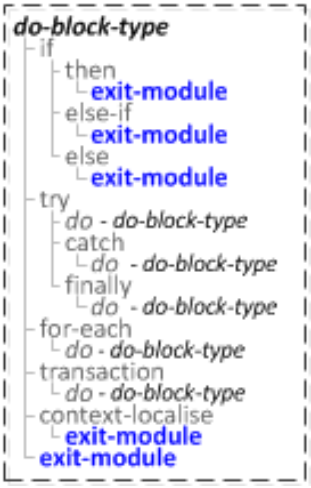
```
<fm:eval expr="concat('<assign to=', :{root}/ELEMENT_NAME/text(), ' from=&apos;/ROOT/TEST/&apos;>')"/>
```

## Notes

The 'fm' namespace should **not** be specified on the commands provided to these attributes.

# FOX:Commands:exit-module

## Schema Location


## Description

Removes the current module from the callstack.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| type | String Literal | • exit-this-preserve-callbacks - Default<br>• exit-this-cancel-callbacks<br>• exit-all-cancel-callbacks | No |
| uri | XPath String | If this is specified then FOX redirects the browser to the given URI. | No |

## Examples

```
<fm:exit-module [type="string" uri="XPath"]/>
```

# FOX:Commands:finally

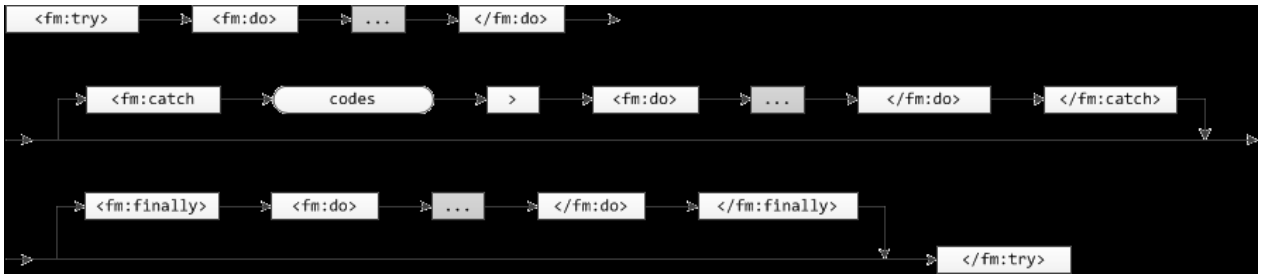
## Schema Location



## Description

Try/catch/finally, similar to Java try/catch/finally blocks.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| code | String Literal | A list of space/tab separated codes to catch, as defined by the code parameter of fm:throw. | Yes |

## Examples

```
<fm:try>
  <fm:do>
    ...
    [command-list]
    ...
  </fm:do>
  [<fm:catch codes="string">
    <fm:do>
      ...
      [command-list]
      ...
    </fm:do>
```

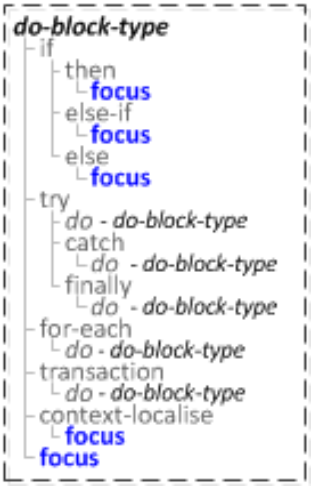
```
    </fm:catch> ...]
    [<fm:finally>
      <fm:do>
        ...
        [command-list]
        ...
      </fm:do>
    </fm:finally>]
  </fm:try>
```

### Notes

See fm:throw for how to raise an error.

# FOX:Commands:focus

## Schema Location


## Description

This is used to focus on the set out DOM element matched by "xpath". Selection start and end can be used to select text within an input field, but both must be specified in order for text to be selected.

## Syntax

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| xpath | Single Node Complex XPath | | Yes |
| selectionStart | Integer | | No |
| selectionEnd | Integer | | No |

## Examples

```
<fm:focus xpath="XPath" [selectionStart="integer" [selectionEnd="integer"]]/>
```

### Focus on an element in the Data DOM

```
<fm:focus xpath="/*/CD_INFO/TRACK_TITLE"/>
```

### Focus on an element in the Theme DOM

```
<fm:focus xpath=":{theme}/ASSIGN_TO/WUA_ID"/>
```

### Use a context pointer to focus

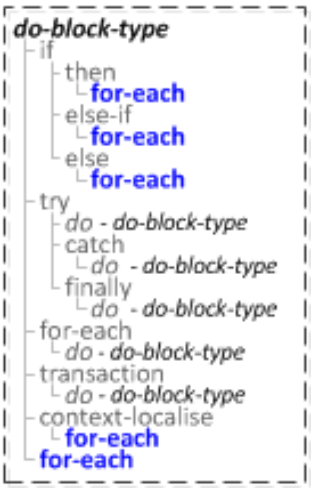
```
<fm:focus xpath=":{action}/FORENAME"/>
```

### Focus on an element and highlight all of its text

```
<fm:focus xpath="/*/CD_INFO/TRACK_TITLE" selectionStart="0" selectionEnd="999999999"/>
```

# FOX:Commands:for-each

## Schema Location

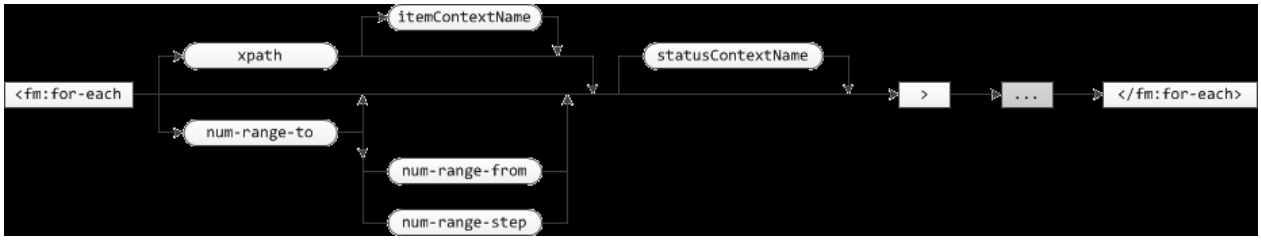

## Description

Provides a method for looping through either:

- A DOM list provided by `xpath`.
- A range of numbers from 0 (by default) to `num-range-to`.

Commands within the for-each command's `<fm:do>` block can refer to the current item in the loop using `itemContextName` (for XPath loops) or `statusContextName` (for either loop type).

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| xpath | Complex XPath | XPath targetting a repeating element. The for-each loop will run for each node in the targeted node list. | No |
| num-range-to | Integer | If not using an XPath, then this is the number to increment to. | No |
| num-range-from | Integer | This is the number to increment to.<br>**Default:** `0`<br>**Note:** Only avaliable when using `num-range-to` for-each loop. | No |
| num-range-step | Integer | This is the number to increment by after each iteration.<br>**Default:** `1`<br>**Note:** Only avaliable when using `num-range-to` for-each loop. | No |

| | | | |
|---|---|---|---|
| itemContextName | String Literal | Name of the context label targeting the current item being processed in the loop. Using "test" here will create :{test} for the scope of the loop.<br>**Default:** loopitem<br>**Note:** Only avaliable when using `xpath` for-each loop. | No |
| statusContextName | String Literal | Name of the context label targeting metadata about the loop. See examples below for further information. Using "test" here will create :{test} for the scope of the loop.<br>**Default:** loopstatus | No |

# Examples

```
<fm:for-each [xpath="XPath" | num-range-to="integer"] [num-range-from="integer" num-range-step="integer" itemContextName="string" statusContextName="string"]>
  <fm:do>
    ...
    [command-list]
    ...
  </fm:do>
</fm:for-each>
```

## Assigning Sequential Numbers To A Node List

You can use 4GL for this. See: fm:assign - Assigning Sequential Numbers To A Node List

## statusContextName Contents For A 5 Node XPath Loop

### 1st iteration

```
<iterator-status>
  <index>0</index>
  <count>1</count>
  <currentStep>0</currentStep>
  <isFirst>true</isFirst>
  <isLast>false</isLast>
  <begin>0</begin>
  <end>4</end>
  <step>1</step>
</iterator-status>
```

### 2nd iteration

```
<iterator-status>
  <index>1</index>
  <count>2</count>
  <currentStep>1</currentStep>
  <isFirst>false</isFirst>
  <isLast>false</isLast>
  <begin>0</begin>
  <end>4</end>
  <step>1</step>
</iterator-status>
```

**3rd iteration**

```
<iterator-status>
  <index>2</index>
  <count>3</count>
  <currentStep>2</currentStep>
  <isFirst>false</isFirst>
  <isLast>false</isLast>
  <begin>0</begin>
  <end>4</end>
  <step>1</step>
</iterator-status>
```

**4th iteration**

```
<iterator-status>
  <index>3</index>
  <count>4</count>
  <currentStep>3</currentStep>
  <isFirst>false</isFirst>
  <isLast>false</isLast>
  <begin>0</begin>
  <end>4</end>
  <step>1</step>
</iterator-status>
```

**5th iteration**

```
<iterator-status>
  <index>4</index>
  <count>5</count>
  <currentStep >4</currentStep>
  <isFirst>false</isFirst>
  <isLast>true</isLast>
  <begin>0</begin>
  <end>4</end>
  <step>1</step>
</iterator-status>
```

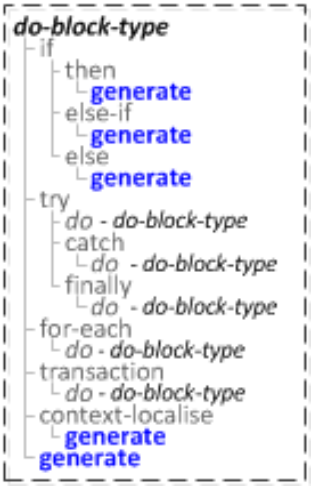
## Notes

`fm:for-each` is inefficient and should be avoided. Most of the functionality provided by for-each is available in the XPath specification.

When used in presentation code then you do not use the `<fm:do>` tags.

# FOX:Commands:generate

## Schema Location


## Syntax

```
<fm:generate method="string" buffer="string" state="string" storage-location="string" url-xpath="XPath" copy-to-xpath="XPath" expires="string" output-type="string" client-cache="string" scope="string"/>
```

## Attribute Summary

| Attribute | Data Type | Description |
|---|---|---|
| method | String Literal | • url (internal)<br>• storage-location<br>• copy-to<br>• preview |
| buffer | String Literal | |
| state | String Literal | |
| storage-location | String Literal | |
| url-xpath | Single Node Complex XPath | Can also use "target", but it is deprecated. |
| copy-to-xpath | Single Node Complex XPath | |
| expires | String Literal | |
| output-type | String Literal | 'CSV', 'XLS' (see Spreadsheet Generation section) or 'XHTML' |
| client-cache | String Literal | |
| scope | String Literal | |

# Examples

## Spreadsheet Generation

CSV and XLS files can be generated from DOM data with the following markup nested within the <fm:generate> command.

```
 <fm:generate method="preview" output-type="XLS">
   <fm:generate-sheet name="'''Literal String'''
row-expr="'''[[FOX:Reference:Complex_XPath|Complex XPath]]'''"
show-headers="'''[[FOX:Reference:Complex_XPath|XPath Boolean]]'''">
     <fm:generate-column name="'''[[FOX:Reference:Complex_XPath|XPath
 String]]'''" column-expr="'''[[FOX:Reference:Complex_XPath|Complex
XPath]]'''" visible-expr="'''[[FOX:Reference:Complex_XPath|XPath
Boolean]]'''"/>
     ...
   </fm:generate-sheet>
 </fm:generate>
```

row-expr is used to select a Node List which represents the rows of the spreadsheet, e.g. ":{root}/CHEMICAL_LIST/CHEMICAL". <fm:generate-column> is then used to select the column values. *column-expr* is evaluated relative to the complex type selected by the *row-expr*, e.g. "./CHEMICAL_NAME". Use *visible-expr* to control the visibility of a column on a sheet by sheet basis.

Note the following considerations:

- visible-expr is evaluated from the attach point of this command's wrapping <fm:do> block, not the row-expr or column-expr node.
- CSV generation shows column headers by default whereas XLS hides them by default.
- Values marked-up in the schema as decimals are truncated to 2 decimal places. If this behaviour is not desired, use concat(ELEMENT_NAME,' ') to force them to be output as a string.
- XLS files may include more than one sheet by specifying multiple <fm:generate-sheet> elements.

# FOX:Commands:go-to-page

## Description

***fm:go-to-page*** is a command used to procedurally change the current page for a pagination result cache.

## Syntax

```
<fm:go-to-page match="[ Root of Result Set" pagination-invoke-name="[ Invoke Name ]" number="[ Page Number ]" out-of-bounds-action="[ Fail Action ]"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | xs:string XPath | Match to root element of ***fm:run-query2*** **result set** | Yes |
| pagination-invoke-name | xs:string | Invoke name of pagination query (as specified by ***fm:run-query2*** ) | Yes |
| number | xs:string | New page number of pagination cache | Yes |
| out-of-bounds-action | xs:string | Name of action to be called when specified number attribute is out of bounds | Np |

## Examples

```
<fm:action name="action-go-to-page-editbox" test:run=".">
  <fm:do>
    <fm:go-to-page match="/ROOT/SEARCH_RESULTS" pagination-invoke-name="xx_invoke" number="string(:{theme}/FORM/PAGE_NUM)" out-of-bounds-action="action-bad-bounds"/>
  </fm:do>
</fm:action>
<fm:action name="action-bad-bounds">
  <fm:do>
    <fm:alert message="Overstepped page boundaries!"/>
  </fm:do>
</fm:action>
```

When calling ***action-go-to-page-editbox*** the pagination cache of the page number located in ***:{theme}/FORM/PAGE_NUM*** will be loaded into the DOM. If the number is out of bounds of the pagination cache, and alert message will trigger.

## Notes

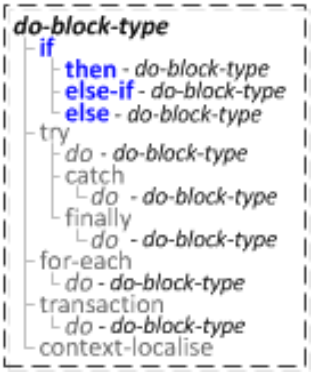
More information for ***fm:go-to-page*** can be found at fm:run-query2

## Related

- fm:run-query2
- fm:page-controls-out

# FOX:Commands:if

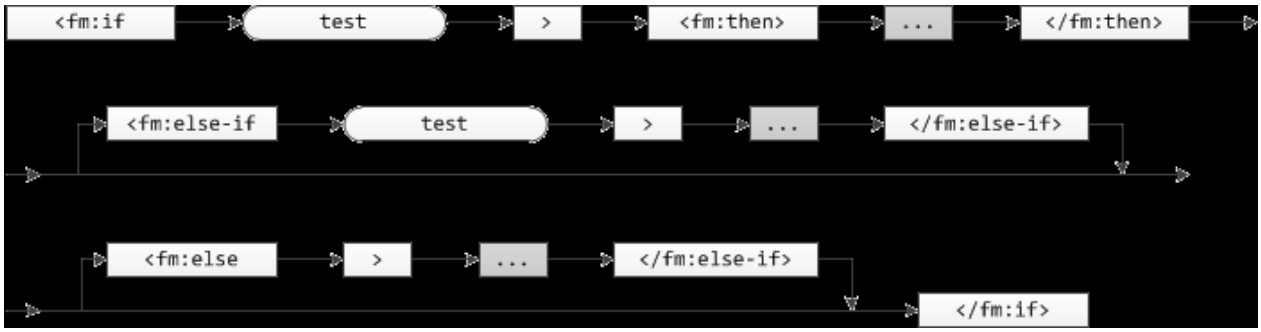
## Schema Location


## Description

Traditional 3GL branching logic. Try to keep usage to a minimum.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| test | XPath Boolean | The XPath expression to test. | Yes |

## Examples

```
<fm:if test="XPath">
  <fm:then>
    ...
    [command-list]
    ...
  </fm:then>
  [<fm:else-if test="XPath">
    ...
    [command-list]
    ...
  </fm:else-if> ...]
```

```
  [<fm:else>
    ...
    [command-list]
    ...
  </fm:else>]
</fm:if>
```

### Presentation logic

```
<fm:if test="/*/ORDER/ORDER_ITEM_LIST/ORDER_ITEM">
  <fm:then>
    <b>There is at least one outstanding order to be processed.</b>
  </fm:then>
  <fm:else>
    There are no outstanding orders.
  </fm:else>
</fm:if>
```

### Action logic

```
<fm:if test=":{action}/QTY > 5">
  <fm:then>
    <fm:alert message="string(concat('Item quantity is: ', :{action}/QTY))"/>
  </fm:then>
  <fm:else>
    <fm:alert message="Quantity is less than 5"/>
  </fm:else>
</fm:if>
```

### Validate logic

```
<fm:validate match=":{theme}/CRITERIA/*" clear="BOTH" check="ALL"/>
<fm:if test="not(:{error}/error-list/fox-error)">
  <fm:then>
    <fm:run-query interface="dbint-search" query="query-search" match=":{theme}/CRITERIA" mode="PURGE-ALL"/>
  </fm:then>
</fm:if>
```

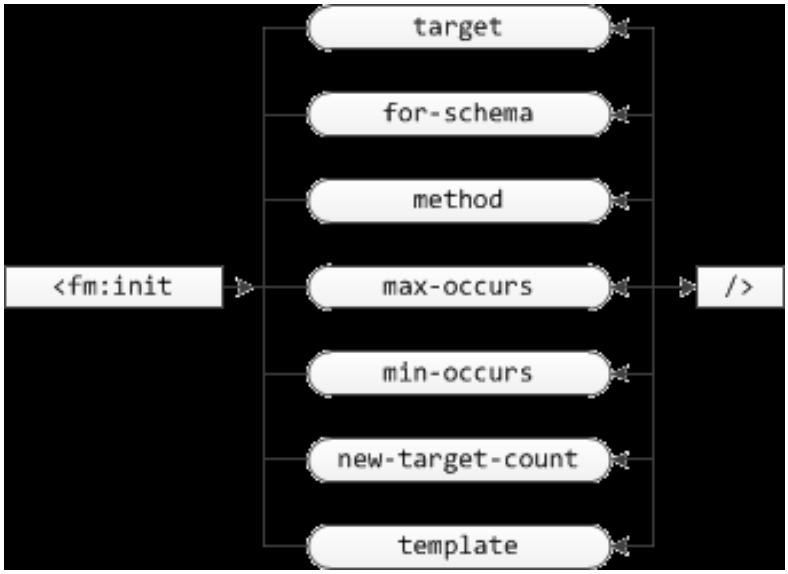
Only run the search query if validation was successful.

# FOX:Commands:init

## Description

Initialises elements in a DOM. You may optionally use the module's schema to determine which elements are initialised.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| target | Complex XPath (with caveat: see Notes) | An XPath specifying the name and location of the element that you want to initialise. Normally pinpoints one or more collection elements (records).<br>**Default:** "." | No |
| for-schema | Simple XPath | This is an XPath referring to the XML schema in the module relative to the *target* and specifies which elements that are specified in the schema should be initialised.<br>**Default:** "." | No |
| method | String Literal | • both - Add elements to both existing and new nodes.<br>• augment - Add elements to nodes that already exist.<br>• new - Add elements only to new nodes.<br>See Notes for a detailed explanation.<br>**Default:** "both" | No |
| max-occurs | XPath Integer | If initialising a repeating element, this is the maximum number of occurrences you want. Takes precedence over the new-target-count parameter. | No |
| min-occurs | XPath Integer | If initialising a repeating element, this is the minimum number of occurrences you want. Takes precedence over the new-target-count parameter. | No |
| new-target-count | XPath Integer | When initialising a repeating element, if you want more than one, set this parameter to the number of elements you want. It is adjusted by the max- and min-occurs parameters if defined. | No |
| template | String Literal | It is possible to create templates in template-list in FOX modules. If you want to initialise your element using a template, specify the name of the template here. The element is initialised as per other parameters and then the contents of the specified template is copied into it. | No |

## Examples

```
<fm:init [target="Xpath" for-schema="XPath" method="string" max-occurs="integer" min-occurs="integer" new-target-count="integer" template="string"]/>
```

## Notes

### Understanding the `fm:init` command

The `fm:init` command is powerful, but can sometimes cause confusion. The `method` attribute determines how the command runs, and is affected by the other attributes.

Consider the following command:

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="new"/>
```

In this command, you are asking FOX for one new PERSON element within a PERSON_LIST element (which may or may not exist). The 'target element' of this command is therefore PERSON. As FOX executes this command, it walks the *target* path and determines if the element specified at each level of the path exists. If the *method* on the command is a 'create' method (i.e. *new* or *both*), and the element does not exist, it is created. The *augment* method does not create new elements based on the *target* attribute.

Now consider this command:

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="new" new-target-count="5"/>
```

This is now asking for 5 new PERSON elements, again within a PERSON_LIST. The command will create at most one PERSON_LIST, if it does not exist, and 5 PERSON elements underneath it. If it finds more than 1 PERSON_LIST elements when it executes, each PERSON_LIST will be populated with 5 PERSON elements. (If you want to create multiple PERSON_LIST elements you must do this in a separate init command.)

You can specify *min-occurs* and *max-occurs* to influence the behaviour of *new-target-count*. These attributes enforce a minimum or maximum number of 'target elements' which are created. The arithmetic for these calculations is relative to the parent of the 'target element'. In these examples, the parent of the 'target element' is PERSON_LIST. Therefore in the above example, each PERSON_LIST will have 5 PERSON records initialised within it. Therefore they are NOT global to the command (i.e. if 3 PERSON_LISTs exists, the example will actually created 15 new PERSON records within the :{theme} DOM).

The *both* method behaves slightly differently to *new*:

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="both"/>
```

This will only create a PERSON element within PERSON_LIST if one does not already exist. It will still create a PERSON_LIST if necessary. You may however specify a *new-target-count* or *min-occurs* value. This forces the *both* method to create new PERSON elements (providing *max-occurs* is not violated).

The *augment* method is useful in conjunction with *for-schema*. Whilst it will not create any new 'target elements' (i.e. PERSON elements), it will initialise elements **within** a 'target element', based on the *for-schema* path. Assume that the module running this command has a valid schema definition for :{theme}/PERSON_LIST/PERSON (providing the elements NAME and DOB):

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="augment" for-schema="./*"/>
```

This will initialise NAME and DOB elements for every existing PERSON, but will not create any new PERSON elements. Additionally, it will not create a PERSON_LIST if it does not exist.

Likewise, this command will create 3 new PERSON elements, then initialise NAME and DOB for both the new PERSON elements it just created and any which already existed:

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="both" for-schema="./*" new-target-count="3"/>
```

Remember, if *new-target-count* is not specified, one 'target element' is created, but only if none exist already.

This behaviour can be adjusted further with the *new* method:

```
<fm:init target=":{theme}/PERSON_LIST/PERSON" method="new" for-schema="./*" new-target-count="3"/>
```

In this case, the NAME and DOB elements will only be initialised for the PERSON elements which were created by the command. Any existing PERSON elements will not have NAME and DOB elements initialised.

## The `target` XPath

The documentation defines the *target* attribute as a Complex XPath, but this is not strictly true. The path may use FOX contexts, but predicates and axes may be invalid. Wildcard operators (such as // or *) are never valid.

The *target* **MUST** be a deterministic path to the desired target element. Using a wildcard such as * is ambiguous, so it is not allowed.

To use predicates in a *target* path, you must be sure that any predicated element exists before running the command. Otherwise FOX will naively attempt to create an element whose name includes the predicate string. For example:

```
<fm:init target=":{theme}/PERSON_LIST[PERSON]/PERSON_EXISTS_FLAG"/>
```

This command is attempting to create a PERSON_EXISTS_FLAG in any PERSON_LIST which contains a PERSON. The command will be successful if at least one PERSON_LIST contains a PERSON. However, if no PERSON_LIST contains a PERSON, or no PERSON_LIST exists, the command will fail and the user will see an error message.

This behaviour may change in a future release.

## Initial Values

You can mark up elements on your module schema to specify how to initialise values within them. These rules are **ONLY** used when the element in question has been initialised by the *for-schema* attribute of the init command. Elements initialised by the *target* attribute are not given initial values.

The init command is aware of the following schema markup. See their pages for more information:

- default - allows a fixed string (XML Schema compliant)
- fixed - as above
- init-xpath - specify an XPath String to initialise the text content of the element. The XPath is relative to the target element.
- init-db-interface and init-query - specifies a query to run from the given db-interface, relative to the target element. Query results are inserted as XML.

### Efficient inits

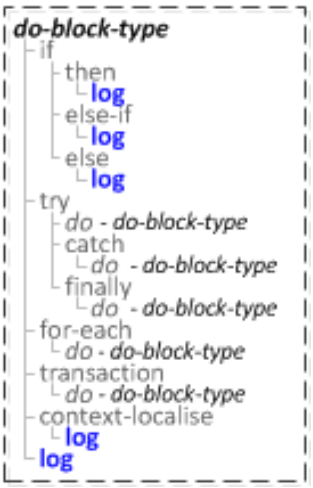
A word about initialising efficiently. If you want to initialise several elements that are not siblings and exist in the schema, you can use the pipe, "|". For example, instead of doing `<fox:init target="/*/CAT"/>` and then `<fox:init target="/*/DOG"/>` you can do `<fox:init target="/*" for-schema="CAT | DOG"/>`.

### Do not initialise too much

A common mistake that people make is to initialise too much. Consider the following *for-schema* XPath: `/*/SECTION_A//*`. This will initialise the whole of section A (with all descendants) when perhaps only a small part of it was needed.

# FOX:Commands:log

## Schema Location


## Description

Logs the given message to the FOX log file.

## Syntax


## Attribute Summary

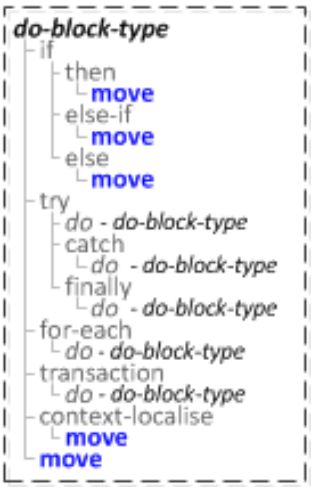
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| message | String Literal | The message to add to the FOX log file. | Yes |

### Examples

```
<fm:log message="string"/>
```

# FOX:Commands:move

## Schema Location


## Description

Moves all targeted source nodes from `from` nodes to `to` nodes.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| from | Complex XPath | The source node(s) to copy.<br>**Default:** "." | No |
| to | Complex XPath | The destination of the node(s) to copy.<br>**Default:** "." | No |

## Examples

```
<fm:move [from="XPath"] [to="XPath"]/>
```

Using following XML Structure:

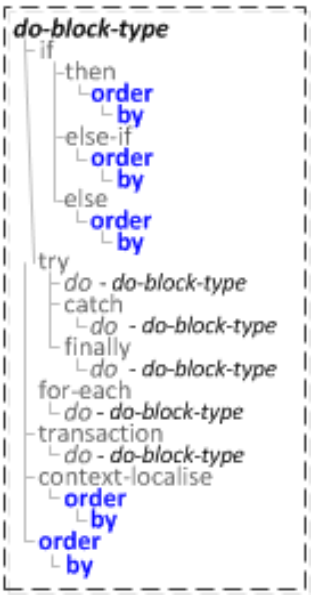
```
<PARENT>
  <NAME>Bob</NAME>
  <AGE>34</AGE>
  <SEX>M</SEX>
  <CHILDREN>
    <CHILD>
      <NAME>Lucy</NAME>
      <AGE>7</AGE>
      <SEX>F</SEX>
    </CHILD>
    <CHILD>
      <NAME>Daniel</NAME>
      <AGE>12</AGE>
      <SEX>M</SEX>
    </CHILD>
    <CHILD>
      <NAME>Nicholas</NAME>
      <AGE>10</AGE>
      <SEX>M</SEX>
    </CHILD>
  </CHILDREN>
</PARENT>
```

```
<fm:move from="/*/CHILDREN" to=":{theme}"/>
```

This would move the entire CHILDREN list from the data DOM in the theme DOM (the CHILDREN list will no longer exist in the data DOM)

# FOX:Commands:order

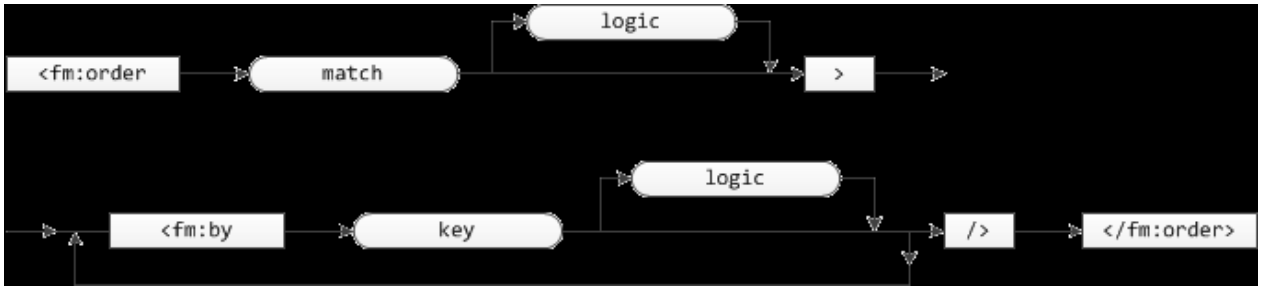
## Schema Location


## Description

Orders the matched node list using the by key attribute according to the specified logic.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | Complex XPath | A list of nodes to order. | No |
| logic | String Literal | • `blank-number-alpha-ascend` - Nulls first, then in lexicographical order [1]<br>• `blank-number-alpha-descend` - Reverse order of blank-number-alpha-ascend.<br>The method to use in ordering the node list, when it is on the fm:order element it is used as the default for any `fm:by` keys without an explicit override.<br>**Default:** `"blank-number-alpha-ascend"` | No |
| key | Single Node Complex XPath | An xpath specifying a node whose value should be sorted by. | Yes |

## Examples

```
<fm:order [match="XPath" logic="string"]>
  <fm:by key="" [logic="string"]/>
  [...]
</fm:order>
```

```
<fm:order match=":{theme}/FILE_LIST/FILE">
  <fm:by key="FILE_ORDER"/>
</fm:order>
```

## Related

- fm:by

# FOX:Commands:parse-spreadsheet

## Description

Turns an uploaded excel spreadsheet into a workbook XML format.

## Attribute Summary

| Attribute | Data Type | Description |
|---|---|---|
| file-storage-location | String Literal | |
| attach | Single Node Complex XPath | |
| storage-location | String Literal | |
| preserve-empty-cells | String Literal | "true" or "false" |

## Examples

```
<fm:storage-location name="sl-DAFT_UPLOADS-parse">
  <fm:cache-key string="sl-DAFT_UPLOADS-parse :1">
     <fm:using using-type="XPATH">:{theme}/PARSED_XML_ID</fm:using>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>WORKBOOK</fm:root-element>
  </fm:new-document>
  <fm:database>
    <fm:query>
      <fm:sql>
SELECT parsed_xml_data
FROM hqmgr.dept_uploads
WHERE id = :1
FOR UPDATE OF parsed_xml_data
      </fm:sql>
      <fm:using>ID</fm:using>
```

```
    </fm:query>
    <fm:update>
      <fm:sql>
UPDATE hqmgr.dept_uploads
SET id = id
WHERE id = :1
      </fm:sql>
      <fm:using>ID</fm:using>
     </fm:update>
   </fm:database>
</fm:storage-location>

...

<fm:parse-spreadsheet file-storage-location="sl-dept-upload" attach=":{action}/../FILE_INPUT"
                      storage-location="sl-DAFT_UPLOADS-parse"
preserve-empty-cells="true">
  <fm:worksheet-list/>
</fm:parse-spreadsheet>
```

## Warnings

- Columns where the visible value is a result of a formula won't get read properly. (These seem to get translated to '#N/A'.) If you need to use a formula column, copy the entire column, then "Paste Special" it back in as "Values".
- Complicated formatting won't get read properly. If you try to resave the file as an xsl and it complains about any kind of loss of fidelty or functionality, this is probably a sign that parse-spreadsheet won't deal well with it.
- Be aware that dates (in date format) are stored as numbers. Dates stored as strings will stay as strings.
- Column headers will be extracted too so if you're going to work on or transform the data make sure you account for them.

# FOX:Commands:part

## Description

***fm:part*** is a child of ***fm:send*** and is used to specify content of a 'part' of a email message (header,footer,conclusion etc).

## Syntax

```
<fm:part section="[ Section Name ]" url-xpath="[ XPath to Section ]"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| section | String Literal | The name of the message section, "body" is special and you can only have one. | Yes |
| url-xpath | Single Node Complex XPath | | Yes |

## Examples

```
<fm:part section="main-page" url-xpath=":{temp}/MAIN_TEXT"/>
<fm:part section="body" url-xpath=":{temp}/BODY"/>
```

The above example will create two sections in a single email, with their content loaded from different areas of the DOM.

## Notes

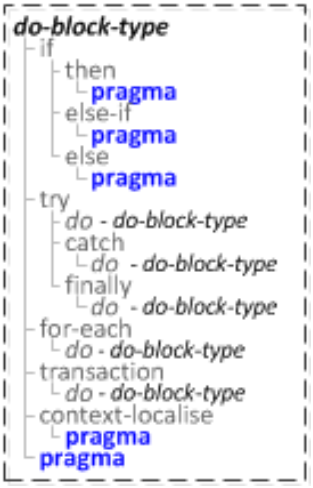
Additional information can be found at fm:send

## Related

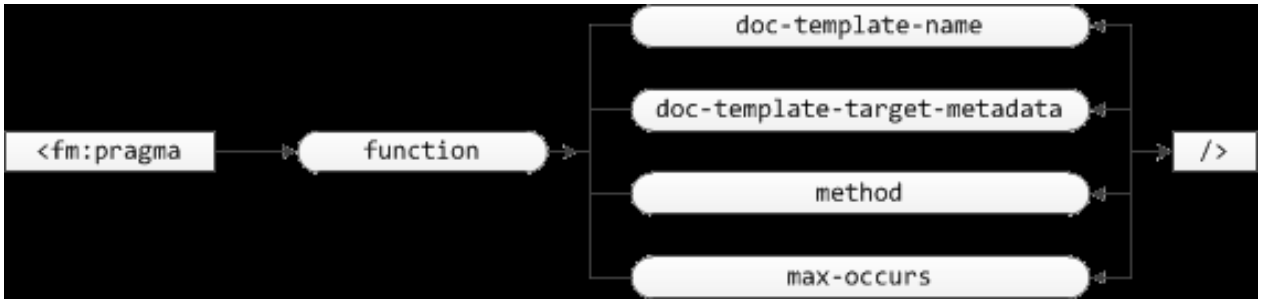
- fm:send

# FOX:Commands:pragma

## Schema Location


## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| function | String Literal | • `flush-applications` - Allows no additional arguments, flushes application cache.<br>• `load-document-template` - Requires `"doc-template-name"` and `"doc-template-target-metadata"`, generates a pdf based on the arguments.<br>• `dump-cached-apps` - Requires `"target"`, dumps a list of everything in the application cache to the targeted node.<br>• `dump-memory-doms` - Requires `"target"`, dumps a list of all cached memory DOMs.<br>• `save-document-changes` - Requires `"doc-template-target-metadata"`.<br>• `set-html-response` - Requires `"target"`, outputs the given target node as HTML.<br>• `set-xml-response` - Requires `"target"` and `"response-mime-type"`, outputs the given target node as XML with the given response-mime-type.<br>• `remote-action-call` - Requires `"target"`, runs a remote action based on the the THREAD_REF, ACTION and [CONTEXT_PATH] nodes of the given target. | Yes |
| doc-template-name | XPath String | The name of the document template to use in generating the pdf. | No |

| | | | |
|---|---|---|---|
| doc-template-target-metadata | Single Node<br>Complex<br>XPath | The node to put the processed template metadata into. | No |
| target | Single Node<br>Complex<br>XPath | A target node for the given function to work with. | No |
| response-mime-type | String Literal | The mime-type to send the response with. | No |

### Examples

```
<fm:pragma function="string" [doc-template-name="XPath" doc-template-target-metadata="XPath" target="XPath" response-mime-type="string"]/>
```

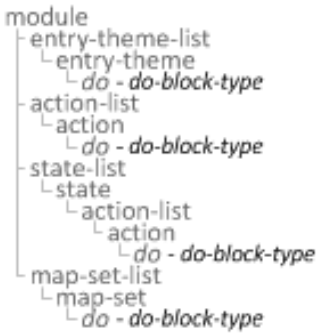
# FOX:Commands:process-set

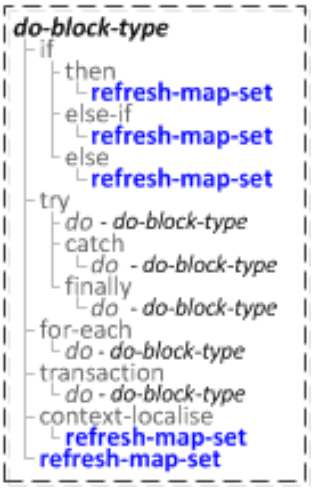
### Description

***fm:process-set*** is a deprecated command and ignored by the FOX Engine as of the latest release.

# FOX:Commands:refresh-map-set

### Schema Location



## Description

Refreshes a map-set based upon its name.

## Syntax


## Attribute Summary

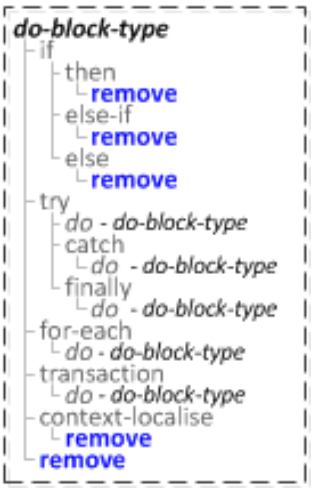
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String Literal | Name of the map-set to refresh. | Yes |

## Examples

```
<fm:refresh-map-set name="string"/>
```

# FOX:Commands:remove

## Schema Location


## Description

Removes all nodes matched by the match xpath.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | Complex XPath | XPath targeting node(s) to remove. | Yes |

### Examples

```
<fm:remove match="XPath"/>
```

#### Remove a whole leg

```
<fm:remove match="/*/NODE"/>
```

#### Remove specific elements under the root node

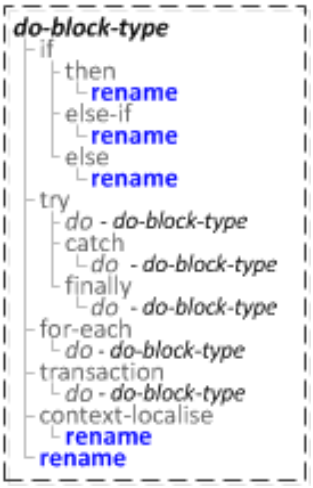
```
<fm:remove match="/*/NODE1 | /*/FAC_NODE2 | /*/NODE3"/>
```

#### Remove all NODE's where SELECT_FLAG is true [relative XPath]

```
<fm:remove match="NODE_LIST/NODE[SELECT_FLAG='true']"/>
```

# FOX:Commands:rename

## Schema Location


## Description

All targeted elements are renamed – content remains unchanged.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | Complex XPath | The node(s) to rename. | Yes |
| rename-to | String Literal | The new name for the nodes. | Yes |

## Examples

```
<fm:rename match="XPath" rename-to="string"/>
```

Using following XML Structure:

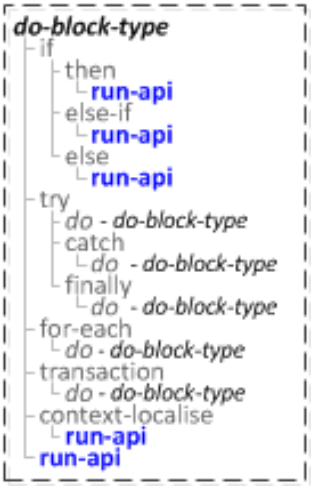
```
<PARENT>
  <NAME>Bob</NAME>
  <AGE>34</AGE>
  <SEX>M</SEX>
  <CHILDREN>
    <CHILD>
      <NAME>Lucy</NAME>
      <AGE>7</AGE>
      <SEX>F</SEX>
    </CHILD>
    <CHILD>
      <NAME>Daniel</NAME>
      <AGE>12</AGE>
      <SEX>M</SEX>
    </CHILD>
    <CHILD>
      <NAME>Nicholas</NAME>
      <AGE>10</AGE>
      <SEX>M</SEX>
    </CHILD>
  </CHILDREN>
</PARENT>
```

```
<fm:rename match="/*/CHILDREN/CHILD/SEX" rename-to="GENDER"/>
```

This will rename the SEX element in all the CHILD nodes to GENDER.

# FOX:Commands:run-api

## Schema Location


## Description

Runs the specified API for every node returned by the match xpath.

## Syntax


## Attribute Summary

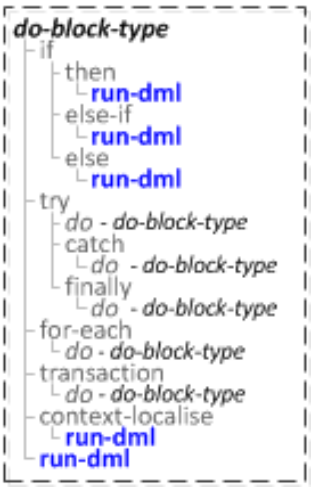
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| interface | String Literal | The name of the database interface to use. | Yes |
| api | String Literal | The name of the api to use within the specified database interface. | Yes |
| match | Complex XPath | The list of nodes to run the query against.<br>**Default:** "." | No |
| mode | String Literal | • `ADD-TO` - Selected items are appended to the DOM regardless of what already exists.<br>• `AUGMENT` - Brings all information from the SELECT statement into the module. Existing records (those of matching key value) are refreshed if different to queried records (new columns can be added), and if key of selected item doesn't exist in the DOM then a new entry will be created.<br>• `PURGE-ALL` - All existing target records are removed and then replaced with the results the select statement gives.<br>• `PURGE-SELECTED` - Fetched records only are removed from the data DOM first, then replaced with values the query returns (based on specified key).<br>**Default:** "`ADD-TO`" | No |

### Examples

```
<fm:run-api interface="string" api="string" [match="XPath"] [mode="string"]/>
```

# FOX:Commands:run-dml

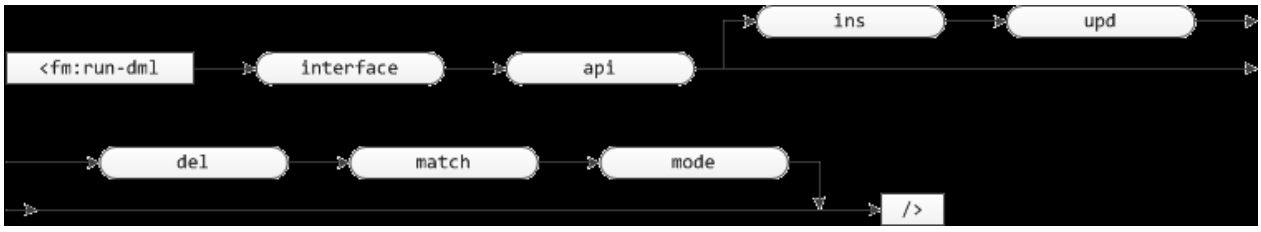
## Schema Location


## Description

Runs a DML statement based upon an `fm:db-interface` `fm:table` definition and the schema definition for the columns.

## Syntax




## Attribute Summary

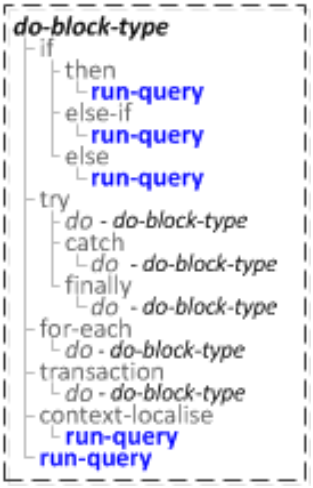
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| interface | String Literal | The name of the database interface with a `fm:table` to use. | Yes |
| api | String Literal | The name of the api to run | Yes |
| ins | String Literal | • `Y`<br>• `N`<br>Whether to run an insert statement.<br>**Default:** `N` | No |
| upd | String Literal | • `Y`<br>• `N`<br>Whether to run an update statement.<br>**Default:** `N` | No |
| del | String Literal | • `Y`<br>• `N`<br>Whether to run an delete statement.<br>**Default:** `N` | No |
| match | Complex XPath | The list of nodes to run the query against.<br>**Default:** "." | No |
| mode | String Literal | • `ADD-TO` - Selected items are appended to the DOM regardless of what already exists.<br>• `AUGMENT` - Brings all information from the SELECT statement into the module. Existing records (those of matching key value) are refreshed if different to queried records (new columns can be added), and if key of selected item doesn't exist in the DOM then a new entry will be created.<br>• `PURGE-ALL` - All existing target records are removed and then replaced with the results the select statement gives.<br>• `PURGE-SELECTED` - Fetched records only are removed from the data DOM first, then replaced with values the query returns (based on specified key).<br>**Default:** `"ADD-TO"` | No |

# Examples

```
<fm:run-dml interface="string" api="string" [ins="string" upd="string" del="string" match="XPath" mode="string"]/>
```

# FOX:Commands:run-query

## Schema Location


## Description

Runs the specified query for every node returned by the match xpath.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| interface | String Literal | The name of the database interface to use. | Yes |
| query | String Literal | The name of the query to use within the specified database interface. | Yes |
| match | Complex XPath | The list of nodes to run the query against.<br>**Default:** "." | No |
| mode | String Literal | • `ADD-TO` - Selected items are appended to the DOM regardless of what already exists.<br>• `AUGMENT` - Brings all information from the SELECT statement into the module. Existing records (those of matching key value) are refreshed if different to queried records (new columns can be added), and if key of selected item doesn't exist in the DOM then a new entry will be created.<br>• `PURGE-ALL` - All existing target records are removed and then replaced with the results the select statement gives.<br>• `PURGE-SELECTED` - Fetched records only are removed from the data DOM first, then replaced with values the query returns (based on specified key).<br>**Default:** `"ADD-TO"` | No |

### Examples

```
<fm:run-query interface="string" query="string" [match="XPath"] [mode="string"]/>
```

# FOX:Commands:run-query2

## Description

***fm:run-query2*** is a new implementation of ***fm:run-query*** with FOX Engine based pagination built in, this replaces the mixed methods of manual pagination either in the query or via the DOM, both of these methods are not ideal due to the overhead involved in reprocessing a large DOM or running the entire query again and filtering results by row number. The run-query2 enhancements fix this by splitting the results of a query up into chunks, storing those chunks in ***foxmgr.pagination_cache*** . Using the ***invoke*** name, these results can be returned one page at a time with minimal overhead.

## Syntax

```
<fm:run-query2
    interface="[ Database Interface ]"
    query="[ Query Name ]"
    match="[ Match XPath ]"
    mode="[ Query Mode ]"
    pagination-definition="[ Pagination Definition Name ]"
    pagination-invoke-name="[ Pagination Invoke Name ]"
    page-controls-position="[ Page Control Position ]"
/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| interface | xs:string | Name of databse interface to tagert | Yes |
| query | xs:string | Name of query in specified database interfrace | Yes |
| match | XPath<br>xs:string | XPath location to match & run query from. | Yes |
| mode | xs:string | Query mode.<br>• PURGE-ALL<br>• PURGE-SELECTED<br>• AUGMENT<br>• ADD-TO | No |
| pagination-definition | xs:string | Name of pagination definition to use, if not specified will use pagination defenition locaiton inside query | No |
| pagination-invoke-name | xs:string | Indentifier for other pagination controls and features to indentify this query with | Yes |
| page-controls-position | xs:string | • Both<br>• Top<br>• Bottom | No |

# Pagination

## Introduction

A basic mechanism for paging of DOM data (such as search results) is implemented in FOXopen (initial implementation available from 4.04.12-15 onwards).

The FOX engine caches results in DOM form in the database and handles switching pages in and out of memory. A Pager (instance of an interface to a set of cached paginated data) is uniquely defined by a key comprising of:

- Pager Type - multiple paging methods can be implemented. Currently you can either page the result-set of a database query or the data of a fm:for-each command (used for example to manually set out some data)
- Invoke Name - A unique invocation name
- Match Fox ID - when querying and setting out data this is the foxid attribute of the element matched.

A database pager relies on both the invoke name and match ID to be unique - if your run-query2 command provided an invoke name but matched multiple nodes, each having a different result-set then FOX would be unable to obtain a handle for each set. Without relying on a dynamic invoke name, multiple accesses to a screen would share the same cached result set without the match foxid.

Basic pagers rely only on the invoke name.

## Concepts

### The Pagination Definition

A basic pagination definition consists of a name and the (currently fixed) size of a page:

```
<fm:pagination-definition-list>
  <fm:pagination-definition name="pd-results-small">
    <fm:page-size>10</fm:page-size>
  </fm:pagination-definition>
</fm:pagination-definition-list>
```

### Linking a Query

The query you desire to be paginated can be marked up with the name of your definition (alternatively you can specify the definition when running the query):

```
<fm:query name="qry-populate-results">
  <fm:target-path match="RESULT"/>
  <fm:primary>
    <fm:key>ID</fm:key>
  </fm:primary>
  <fm:pagination-definition>pd-results-small</fm:pagination-definition>
  <fm:select>
  ...
  </fm:select>
</fm:query>
```

## Running a Paginated Query

Paginated queries must be run using fm:run-query2. NOTE: You should match on the list container element and specify a target path which is the name of the repeating element. This is because the node you match on when running the query must match the node you set out from.

```
<fm:run-query2 interface="db-result-search" query="qry-populate-results" match="/ROOT/SEARCH_RESULTS" mode="ADD-TO" pagination-invoke-name="xx_invoke"/>
```

Whenever you invoke a paginated query, the number of results retrieved, page size, total row count and total page count are included in the :{sys} DOM as below (from FOXr4.04.41 onwards):

```
<sys>
  <sqlquery>
    <rowcount>99</rowcount>
    <paging>
      <totalrowcount>99</totalrowcount>
      <retrievedrowcount>10</retrievedrowcount>
      <pagecount>10</pagecount>
      <pagesize>10</pagesize>
    </paging>
  </sqlquery>
</sys>
```

It is worth noting that when using run-query2 the query definition that is being run must have the "using" elements specified with all attributes, such as: -

```
<fm:using name=":oper_name" sql-type="varchar" datadom-type="xs:string" datadom-location=":{theme}/SEARCH_CRITERIA/OPERATOR_NAME"/>
```

## Setting Out Paginated Results

```
<fm:set-out match="/ROOT/SEARCH_RESULTS" results:mode="." pagination-invoke-name="xx_invoke"/>
```

Note: The set-out should match the same node your query did and specify the same invoke name.

## Page Controls

By default, setting out a paged result set includes page controls both above and below the data. This can be changed with the *page-controls-position* attribute.

If you want greater control over the presentation, you can manually output the controls in your buffer:

```
<fm:page-controls-out invoke-name="xx_invoke" match=":{root}/SEARCH_RESULTS"/>
```

# Notes

## Resetting / Clearing Paginated Results

The pager handle is tied to the foxid of the matched node. If your paginated query matched :{theme}/RESULT_LIST and you attempted to reset the search by doing <fm:remove match=":{theme}/RESULT_LIST/*"/> you may find your results still exist and are output. To force a new pager handle and fresh result set cache you must remove the element that was matched and re-initialise, giving you a new foxid.

## Styling

There is currently some limited styling support for page controls. For example:

```
.page-size {
  font-size: 1.2em;
  font-weight: bold;
  padding-right: 10px;
}

.page-list-prompt {
  font-size: 1.2em;
}

.page-list-number {
  padding: 0 5px;
}

.page-list-number-current {
  padding: 0 5px;
  font-weight: bold;
}
```

## Procedurally Changing Page

You can change the page of a paginated result set procedurall using the go-to-page command. You must supply a match, invoke name and number that you wish to page to. The number attribute is stringifiable. You can optionally specify an out-of-bounds-action to trigger if an attempt to overstep page boundaries is made, for example if the command runs based on user input.

```
<fm:action name="action-go-to-page-editbox" test:run=".">
  <fm:do>
    <fm:go-to-page match="/ROOT/SEARCH_RESULTS" pagination-invoke-name="xx_invoke" number="string(:{theme}/FORM/PAGE_NUM)" out-of-bounds-action="action-bad-bounds"/>
  </fm:do>
</fm:action>
<fm:action name="action-bad-bounds">
  <fm:do>
    <fm:alert message="Overstepped page boundaries!"/>
  </fm:do>
</fm:action>
```

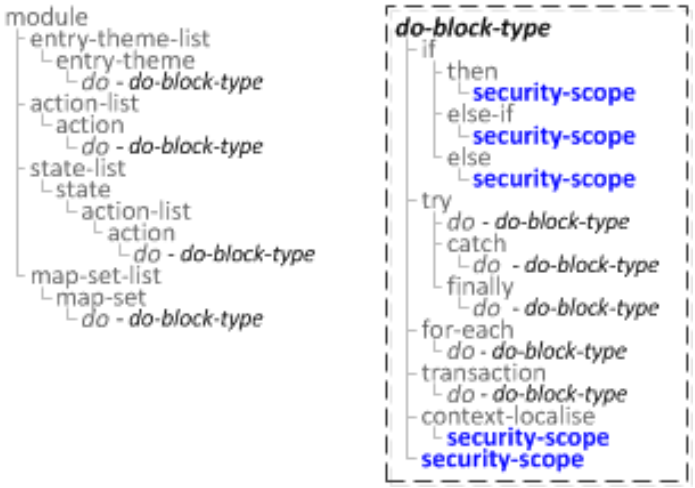
**Note:** Prior to FOXr4.04.41, there is a bug that prevents fm:go-to-page retrieving uncommitted results. From FOXr4.04.41 and onwards, you can run an fm:go-to-page command immediately after the fm:run-query2 command, and you will receive the right results. Issuing a commit after your fm:run-query2 will not work around this problem, as the pagination cache is written using the internal FOX thread connection (which you should not attempt to commit manually). If you wish to use this functionality, it is best to upgrade to FOXr4.04.41.

## Related

- fm:pagination-definition
- fm:go-to-page
- fm:page-controls-out
- FOX Engine Pagination (original source for page)

# FOX:Commands:security-scope

## Schema Location




## Description

Sets the security-scope for the current module.

## Syntax


## Attribute Summary

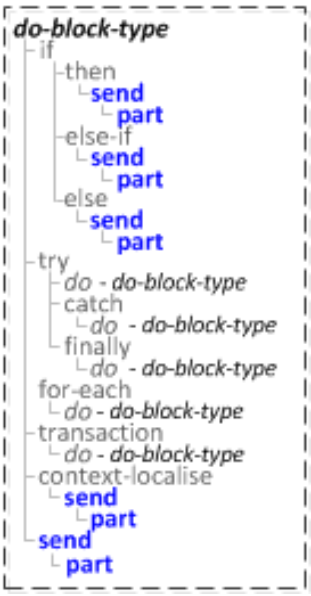
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| csv-system-privs | String Literal | | Yes |
| uref-xpath | Complex XPath | If this xpath returns a string then that is used, if it returns a node list then it is turned into a comma-separated list of values and used. | Yes |
| csv-object-privs | String Literal | | Yes |
| csv-uref-types | String Literal | | Yes |

### Examples

```
<fm:security-scope csv-system-privs="string" uref-xpath="XPath" csv-object-privs="string" csv-uref-types="string"/>
```

# FOX:Commands:send

## Schema Location


## Description

Sends an email.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| addressee-xpath | Single Node Complex XPath | <SEND><br><SUBJECT/><br><PRIORITY/><br><SENDER><br><NAME/><br><ADDRESSEE><br><NAME/><br><ADDRESS/><br><TYPE/><br><NO_SEND/><br><FORMAT/><br></ADDRESSEE><br></SENDER><br></SEND> | Yes |
| section | String Literal | The name of the message section, "body" is special and you can only have one. | Yes |
| url-xpath | Single Node Complex XPath | | Yes |

## Examples

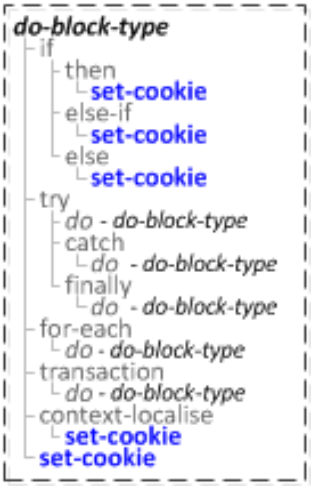
```
<fm:send addressee-xpath="XPath">
  <fm:part section="string" url-xpath="XPath"/>
  [...]
</fm:send>
```

## Related

- fm:part

# FOX:Commands:set-cookie

## Schema Location


## Description

Sets a cookie with the specified name and value.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | XPath String | The name of the cookie. | Yes |
| value | XPath String | The value of the cookie. | Yes |
| max-age | XPath String | The max age for the cookie, in seconds. | Yes |

## Examples

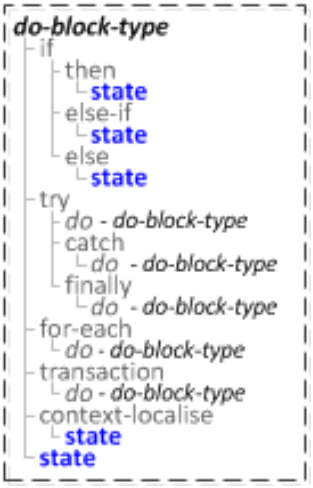
```
<fm:set-cookie name="XPath" value="XPath" max-age="XPath"/>
```

## Notes

Bear in mind that the name/value parameters are xpaths. If you do not wrap static string values with string() then they may well return null or give you unexpected behaviour.

# FOX:Commands:state

## Schema Location


## Description

Does the chosen callstack modification action with the given parameters, should use fm:state-pop, fm:state-push and fm:state-replace instead.

## Syntax


## Attribute Summary

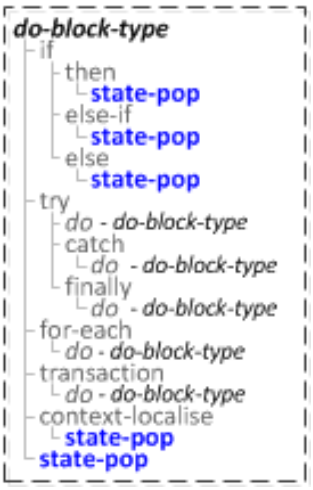
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| action | String Literal | • `pop`<br>• `push`<br>• `replace`<br>**Note:** Only for `action="replace"` and `action="push"`. | Yes |
| name | String Literal | The name of the state to use. | Yes |
| all | String Literal | • `true`<br>• `false`<br>**Default:** `false`<br>**Note:** Only for action="replace".<br>**Note:** When true it replaces all states from the callstack with just the new state. | No |
| attach | Single Node Complex XPath | The attach point for the new state.<br>**Note:** Only for `action="replace"` and `action="push"`. | No |

### Examples

```
<fm:state action="string" name="string" [all="string" attach="XPath"]/>
```

# FOX:Commands:state-pop

## Schema Location


## Description

Pops the current state from the callstack, with an implicit exit-module if it is the last state for the current module.

## Syntax

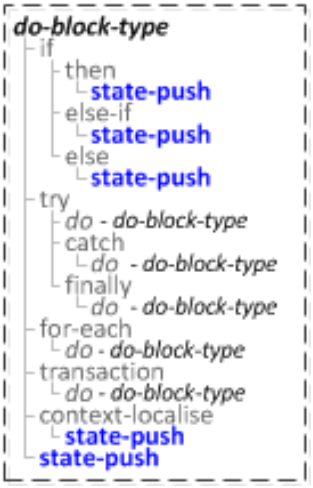
```
<fm:state-pop/>
```

## Examples

```
<fm:state-pop/>
```

# FOX:Commands:state-push

## Schema Location


## Description

Pushes the new state onto the top of the callstack.

## Syntax


## Attribute Summary

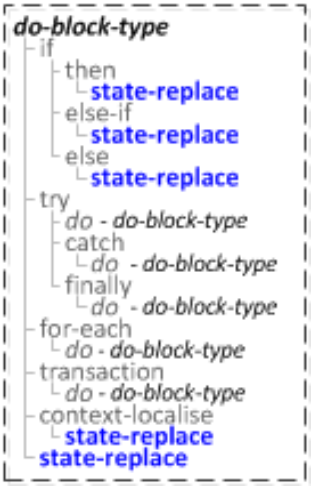
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String Literal | The name of the state to use. | Yes |
| attach | Single Node Complex XPath | The attach point for the new state. | No |

## Examples

```
<fm:state-push name="string" [attach="XPath"]/>
```

# FOX:Commands:state-replace

## Schema Location


## Description

Replaces the current state on the callstack with the new state.

## Syntax


## Attribute Summary

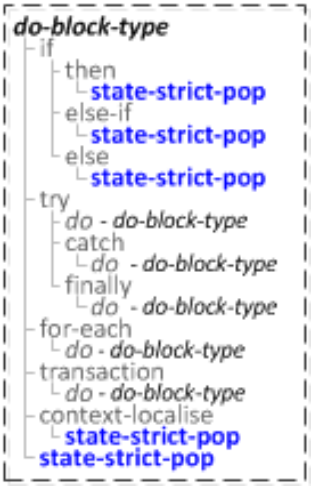
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String Literal | The name of the state to use. | Yes |
| all | String Literal | • `true`<br>• `false`<br>**Default:** `false`<br>**Note:** When true it replaces all states from the callstack with just the new state. | No |
| attach | Single Node Complex XPath | The attach point for the new state. | No |

## Examples

```
<fm:state-replace name="string" [all="string" attach="XPath"]/>
```

# FOX:Commands:state-strict-pop

## Schema Location


## Description

Pops the current state from the callstack, and does not allow an implicit exit-module, instead it will throw an error when there it is the last state of the current module.

## Syntax

```
<fm:state-strict-pop/>
```

## Examples

```
<fm:state-strict-pop/>
```

# FOX:Commands:then

## Description

Commands nested within <fm:then> will be executed if the parent <fm:if> test attribute is evaluated to true.

## Attribute Summary

*None*

## Examples

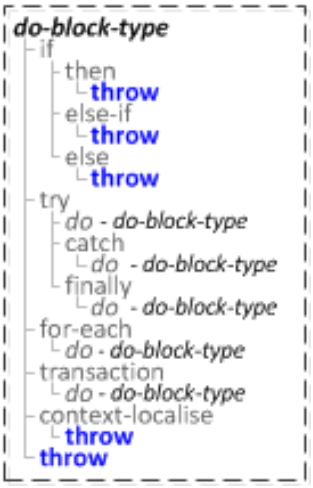
```
<fm:if test="true">
  <fm:then>
    code to execute
  </fm:then>
</fm:if>
```

## Related

- <fm:if>
- <fm:else>
- <fm:else-if>

# FOX:Commands:throw

## Schema Location

## Description

Throws an ExActionFailed exception with the given code and message.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| code | XPath String | This parameter is always converted to upper case and can only contain A-Z, 0-9, - and _ **Special Values**:<br>• `ACTIONIGNORE` - Ignores the rest of the current action.<br>• `ACTIONBREAK` - Jumps straight to auto-action-final/auto-callback-final. | Yes |
| message | XPath String | The error message to display. | Yes |

## Examples

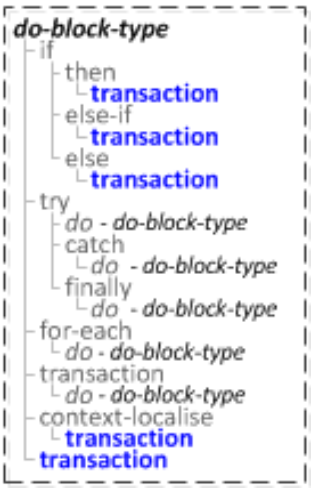
```
<fm:throw code="XPath" message="XPath"/>
```

## Notes

See fm:try for how to catch a raised error.

# FOX:Commands:transaction

## Schema Location

## Syntax


`<fm:do/>` only allowed when operation="JOIN", and required in that case.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| operation | String Literal | • `JOIN` - Posts memory only changes to the storage location (i.e. root DOM) to the database. This allows APIs that reference the storage location base table to work with an up-to-date copy of the DOM.<br>• `COMMIT` - Commit a transaction.<br>• `ROLLBACK` - Rollback a transaction. | Yes |
| transaction | String Literal | • `MAIN` - Not allowed when `operation="JOIN"`.<br>• `STORAGE-LOCATION` - Only allowed when `operation="JOIN"`.<br>**Default:** `"MAIN"` | No |

## Examples

```
<fm:transaction operation="string" [transaction="string"]">
  [<fm:do>
    ...
    [command-list]
    ...
  </fm:do>]
</fm:transaction>
```

## Notes

### General Transactional Boundaries

Transaction JOIN <fm:do> blocks all use the *same* database transaction as each other:

- Each transaction JOIN <fm:do> has access to the uncommitted work of a prior transaction JOINed <fm:do> block
- The transaction exists for the entire of the top level action (aka "page request/response cycle"), so a transaction JOIN in an entry-theme can see uncommitted changed from a calling module.

Transaction JOIN <fm:do> blocks are executed on a *different* database transaction than your outer <fm:do> block.

- A transaction JOIN <fm:do> block does not have access to the uncommitted work that was processed outside of a transaction JOIN <fm:do> block.
- The outer <fm:do> block does not have access to the uncommitted work that was processed inside a transaction JOIN <fm:do> block.

### A warning regarding COMMITs:

- You **must not** issue a COMMIT inside a transaction JOIN (PL/SQL APIs called must be strictly transactional). FOX will automatically commit or rollback the transaction itself.
- If a COMMIT is issued inside a transaction JOIN, FOX will only be able to rollback to that point if an error occurs, and any/all of the thread DOM, data DOM and other data tables may be left in an inconsistent state.

### Avoid DOM manipulation within a transaction JOIN

- The DOM is only posted to the database at the start of a transaction JOIN. Subsequent changes you make within the <fm:do> block itself (i.e. inits, removes) will not be visible within the transaction, so any Xviews, stored procedures, etc will not see these changes.
- Additionally, if you run code in your transaction JOIN which changes the target record of your storage location (i.e. updating the status control fields), any changes you attempt to make to the DOM within or after the transaction JOIN will result in an error.

# FOX:Commands:try

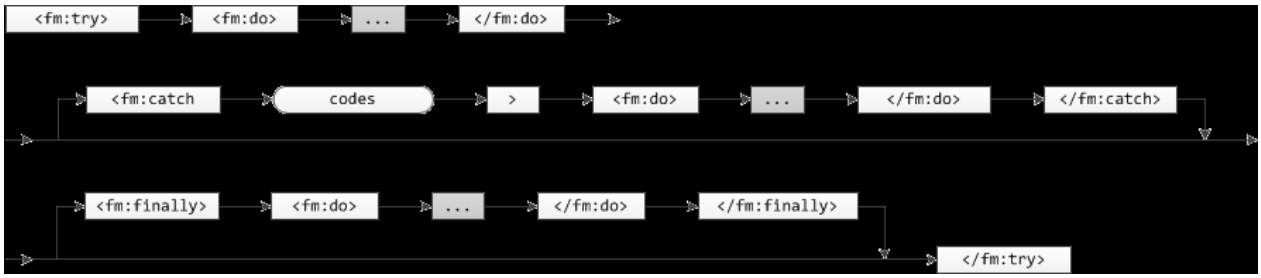
## Schema Location



## Description

Try/catch/finally, similar to Java try/catch/finally blocks.

## Syntax



## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| code | String Literal | A list of space/tab separated codes to catch, as defined by the code parameter of fm:throw. | Yes |

## Examples

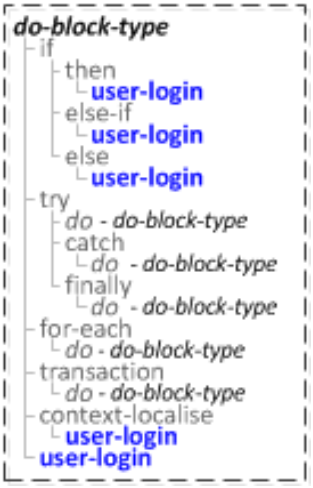
```
<fm:try>
  <fm:do>
    ...
    [command-list]
    ...
  </fm:do>
  [<fm:catch codes="string">
    <fm:do>
      ...
      [command-list]
      ...
    </fm:do>
  </fm:catch> ...]
  [<fm:finally>
    <fm:do>
      ...
      [command-list]
      ...
    </fm:do>
  </fm:finally>]
</fm:try>
```

## Notes

See fm:throw for how to raise an error.

# FOX:Commands:user-login

## Schema Location


## Description

Attempts to login as the given user with the given password.

## Syntax


## Attribute Summary

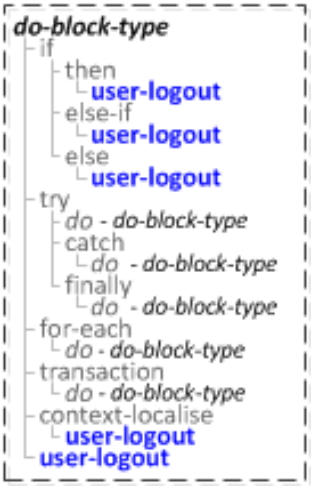
| Attribute | Data Type | Description | Required |
|---|---|---|---|
| wua-login-id | XPath String | The web user account login id to try. | Yes |
| password | XPath String | The account password to try. | Yes |
| status-code | Single Node Complex XPath | A node to place the status code of the login attempt. | No |
| status-message | Single Node Complex XPath | A node to place the status message of the login attempt. | No |

## Examples

```
<fm:user-login wua-login-id="XPath" password="XPath" [status-code="XPath" status-message="XPath"]/>
```

# FOX:Commands:user-logout

## Schema Location


## Description

Logs out the current user.

## Syntax

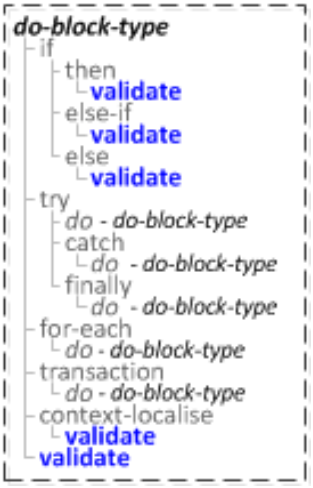
```
<fm:user-logout/>
```

## Examples

```
<fm:user-logout/>
```

# FOX:Commands:validate

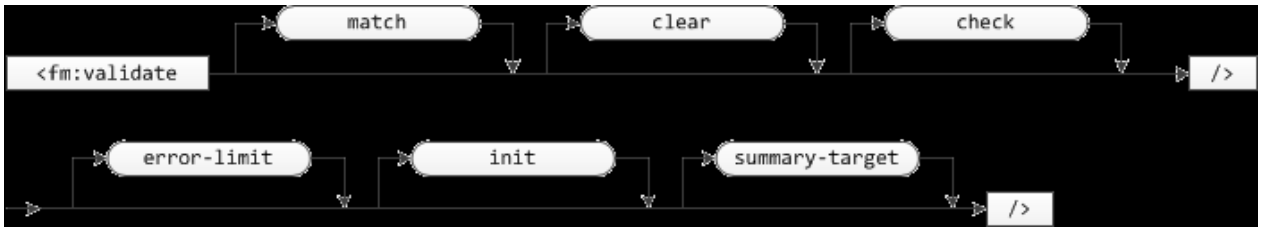
## Schema Location


## Description

Validates the matched nodes against the schema definition.

## Syntax




## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| match | Complex XPath | The list of nodes to validate.<br>**Default:** "." | No |
| clear | String Literal | • `NONE` - Do not clear any errors.<br>• `CLEAR-NODE` - Clear fox-error nodes.<br>• `CLEAR-SUMMARY` - Clear the :{error} DOM.<br>• `BOTH` - Clears both fox-error nodes and the :{error} DOM.<br>**Default:** `"NONE"` | No |
| check | String Literal | • `NONE` - Do not validate at all, used to just clear errors.<br>• `CONTENT` - Check the content of the nodes against the schema rules.<br>• `CARDINALITY` - Check the cardinality of the nodes against the schema rules.<br>• `ALL` - Check both the cardinality and content of the nodes against the schema rules.<br>**Default:** `"ALL"` | No |
| error-limit | Integer | Validation ends after the error-limit is reached.<br>**Default:** `100` | No |

| | | | |
|---|---|---|---|
| init | String Literal | • `Y` - Inits a new blank node to put a fox-error in if the node should exist but doesn't. Doesn't use any initialisation rules specified in the schema - always just creates a blank node to put the `fox-error` into.<br>• `N` - Ignores missing nodes.<br>**Default:** `"Y"` | No |
| summary-target | Simple XPath | Where to place the summary of errors. This is in the context of the current attach point. The default is special in that it points to the :{error} DOM, and is the only way to do so.<br>**Default:** `":{error}/error-list"` | No |

# Supported XSD markup

FOX validation supports a limited subset of XSD markup:

## XSD Datatypes

- xs:string
- xs:decimal
- xs:date
- xs:long
- xs:int
- xs:dateTime
- xs:time
- xs:boolean
- xs:positiveInteger
- xs:negativeInteger
- xs:integer
- xs:anyType

## XSD Facets (where appropriate)

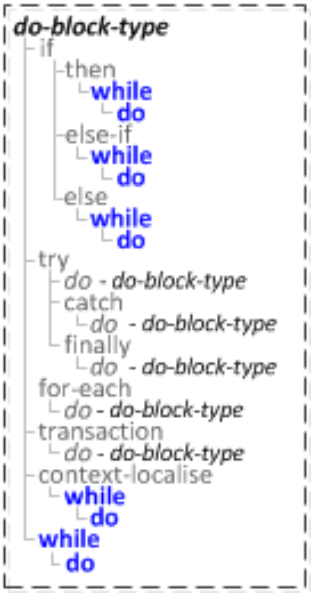
- xs:totalDigits
- xs:fractionDigits
- xs:length
- xs:maxLength
- xs:minLength
- xs:minInclusive
- xs:maxInclusive
- xs:minExclusive
- xs:maxExclusive

# Examples

```
<fm:validate [match="XPath" clear="string" check="string" error-limit="integer" init="string" summary-target="XPath"]>
```

# FOX:Commands:while

## Schema Location


## Description

Executes the commands in `command-list` while the `xpath` expression continues to return `true`.

**NOTE: DO NOT USE THIS COMMAND.** It is unsafe and needs to be modified or deprecated in a future release. `fm:while` does NOT provide any safeguards against infinite loops. Misuse of this command can cause system instability.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| xpath | XPath Boolean | The XPath to evaluate to determine whether the while loop should continue. | Yes |

## Examples

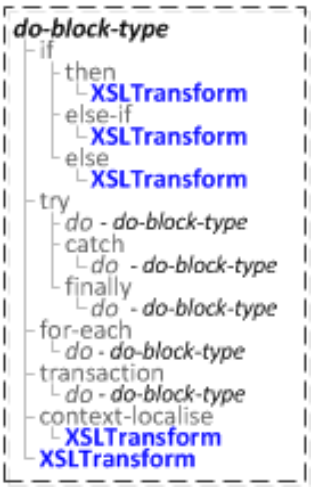
```
<fm:while xpath="XPath">
  <fm:do>
    ...
    [command-list]
    ...
  </fm:do>
</fm:for-each>
```

## Notes

**NOTE: DO NOT USE THIS COMMAND.** It is unsafe and needs to be modified or deprecated in a future release. `fm:while` does NOT provide any safeguards against infinite loops. Misuse of this command can cause system instability.

# FOX:Commands:XSLTransform

## Schema Location


## Description

Transforms the given file-type source node using the given XSL stylesheet and puts the result into the target node.

## Syntax


## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| source-xml-xpath | Single Node Complex XPath | The node to transform. | Yes |
| source-xsl-URI | String Literal | The url of the XSL stylesheet to use (can be a FOX component). | Yes |
| target-xpath | Single Node Complex XPath | The node to put the results into. | Yes |

## Examples

```
<fm:XSLTransform source-xml-xpath="XPath" source-xsl-URI="string" target-xpath="XPath">
```

## Notes

**This command is not recommended for use.** For performance reasons, it is preferable to use the XMLTRANSFORM function provided by Oracle XMLDB (i.e. in a PL/SQL API) to perform XSL Transformations within FOX.