# FOX Module Reference

# FOX:Module:MODULE REFERENCE

- fm:action-list
- fm:action
- fm:api
- fm:application-title
- fm:attr
- fm:authentication
- fm:build-notes
- fm:cache-key
- fm:comments
- fm:control
- fm:css-list
- fm:css
- fm:current-of-item
- fm:data-mapping
- fm:data-type
- fm:database
- fm:db-interface-list
- fm:db-interface
- fm:delete
- fm:description
- fm:display-attr-list
- fm:documentation
- fm:entry-theme-list
- fm:entry-theme
- fm:file-storage-location
- fm:for-each-fetch
- fm:header
- fm:help-text
- fm:history
- fm:insert
- fm:into
- fm:key
- fm:library-list
- fm:library
- fm:lock
- fm:map-path
- fm:map-set-list
- fm:map-set
- fm:matrix-and
- fm:matrix-author
- fm:matrix-container-record-number
- fm:matrix-date-closed
- fm:matrix-date-created
- fm:matrix-date-registered
- fm:matrix-external-reference

- fm:matrix-into
- fm:matrix-notes
- fm:matrix-order-by
- fm:matrix-or
- fm:matrix-record-number
- fm:matrix-record-title
- fm:matrix-record-type
- fm:matrix-search
- fm:mode-rule
- fm:module
- fm:name-space-list
- fm:name-space
- fm:name
- fm:new-document
- fm:page-size
- fm:pagination-definition-list
- fm:pagination-definition
- fm:param-list
- fm:parameter-list
- fm:parameter
- fm:param
- fm:post-page
- fm:pre-condition
- fm:pre-page
- fm:presentation
- fm:primary
- fm:query
- fm:refresh-in-background
- fm:refresh-timeout-mins
- fm:return-list
- fm:return
- fm:root-element
- fm:row-lock
- fm:schema
- fm:security-list
- fm:security-rule
- fm:select
- fm:set-buffer
- fm:set-page
- fm:show-popup
- fm:sql
- fm:state-list
- fm:statement
- fm:state
- fm:storage-location-list
- fm:storage-location
- fm:table

- fm:target-path
- fm:template-list
- fm:template
- fm:title
- fm:update
- fm:using
- fm:version-desc
- fm:version-no
- fm:view-rule
- fm:xml-commit

# FOX:Module:action

## fm:action

### Description

```
<fm:action name="String" [ any-namespace:prompt="XPath String" any-namespace:run="XPath Boolean" any-namespace:ro="XPath Boolean"  ... ] >
  <fm:do>
    [command list]
  </fm:do>
</fm:action>
```

Define an action. Commands in *command list* are executed from top to bottom. See here for a list of possible commands. Commands which could possibly displace the current module from the top of callstack (i.e. state-pop, call-module) must be last in the Command list.

To call an action by name use fm:call. To call an action as a callback action for a module call, set it as the callback-action on the fm:call-module command. You can also assign fox:action and fox:change-action attributes to schema elements.

Actions can be defined at module level or state level. State level actions are accesible at module level, and from different states, by prefixing the action name with the state name, e.g.

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

Display an action on the page using menu-out or action-out.

### Optional Attribute Description

- **fox:actionStyle** - CSS style element.
- **fox:actionClass** - CSS class element.
- **fox:confirm** - Text of a Javascript confirm dialog which appears when the user clicks the widget.
- **any-namespace:displayOrder** - Display order as used by menu-out.
- **stub-overload** - If the Xpath evaluates to true, any libraried in fm:actions with the same name will replace this one.

## Auto actions

Auto actions are special actions which run implicitly at certain predefined points during FOX processing. An auto action can be defined just like a normal action, but named with a special prefix. These prefixes are as follows:

- auto-action-init
- auto-action-final
- auto-state-init
- auto-state-final
- auto-callback-init
- auto-callback-final

Examples of acceptable auto action names:

```
auto-action-init
auto-action-init-my-auto-action
```

**Auto-action** actions are run before (**init**) and after (**final**) any action which has caused the page to post.

**Auto-state** actions are fired as the state in which they are defined is pushed to (**init**) or popped from (**final**). If they are defined at module level, they run when the module is called (before the entry-theme do block) and when it is exited.

**Auto-callback** actions are run before (**init**) and after (**final**) a callback action.

The auto actions fire in the following order:

1. auto-callback-init
2. auto-action-init
3. auto-state-init
4. the action you're calling (i.e. fm:call, or a page link/button click)
5. auto-state-final
6. auto-action-final
7. auto-callback-final

**Note:** auto-state-final actions do not appear to fire on an exit-module.

## Notes

To use an XPath String as the prompt for an action, you must define the *any-namespace:[ro|edit]* attribute in addition to *any-namespace:run*.

## Related

- fm:action-list
- fm:do
- fm:documentation

# FOX:Module:action-list

## Description

Element for containing multiple fm:action elements. This can be specified at module level, or as a part of an fm:state.

## Examples

**Module level**

```
<fm:module>
  <fm:action-list>
    <fm:action name="action-example-1"/>
    <fm:action name="action-example-2"/>
     ...
  </fm:action-list>
</fm:module>
```

**State level**

```
<fm:state>
  <fm:action-list>
    <fm:action name="action-example-1"/>
    <fm:action name="action-example-2"/>
     ...
  </fm:action-list>
</fm:state>
```

## Notes

The existence of this element is mutually exclusive to the element which contains it.

## Related

- fm:module
- fm:action
- fm:state

# FOX:Module:api

## Description

The ***fm:api*** command specifies the API block within an ***fm:db-interface***. When creating a ***fm:db-interface***, you can mix and match both ***fm:query*** and ***fm:api*** blocks, but the '**fm:query** *ones must be declared before the* **fm:api**' ones.

***fm:api*** blocks can be called in a similar way to ***fm:query*** blocks, by using the ***fm:run-api*** command in an ***fm:do*** block.

The ***fm:using*** child element has optional attributes and can contain an XPath. It can be used in the same way as an ***fm:query*** using clause, but it is possible to specify the direction and type of a bind variable.

When specifying a datadom-type, it is customary to use a datadom-location to match an XPath, rather than include an XPath expression as a text node within the ***fm:using*** tag elements. The datadom-type specifies the type of data coming from or going to the location targeted by the XPath expression and the sql-type specifies a value similar to that of Oracle's own datatypes to dictate how Oracle should treat it.

Under most circumstances, FOX should guess the datatypes, but occasionally it is necessary to override this.

## Syntax

```
  <fm:api name="api name">
    <fm:statement>
  PL/SQL or DML statements go here
    </fm:statement>
    <fm:using
       [name="name of bind variable" ]
       [direction="in|out|in out"]
       [datadom-type="date|datetime|dom|string|time" ]
       [datadom-location="XPath expression of local value to bind"]
       [sql-type="clob|date|varchar|xmltype"]>
          XPath Expression (If datadom-location doesn't exist)
    </fm:using>
  </fm:api>
```

## Examples

The *fm:statement* in the *fm:api* can either contain anonymous PL/SQL or DML. As a PL/SQL block with bind variables, it can look as follows:

```
  <fm:api name="api name">
    <fm:statement>
DECLARE
  l_mynum NUMBER(4) := :num;
  l_mystr VARCHAR2(30) := :str;
  l_myxml XMLTYPE;

BEGIN

  SELECT
    XMLELEMENT("MY_DATA"
```

```
  , XMLELEMENT("MY_STRING",l_myStr)
  , XMLELEMENT("MY_NUM",l_myNum)
    )
  INTO l_myxml
  FROM dual;

  :xml_out := l_myxml;

END;
    </fm:statement>
    <fm:using name=":num">/*/SOME_NUMBER</fm:using>
    <fm:using name=":str" direction="in" datadom-type="string"
              datadom-location="/*/SOME_STRING" sql-type="varchar" />
    <fm:using name=":xml_out" direction="out" datadom-type="dom"
              datadom-location="/*/SOME_XML" sql-type="xmltype" />
  </fm:api>
```

The above code will take a Number and String from the Data DOM, binding it to the PL/SQL. The PL/SQL will do some processing, in this case it will generate an XML structure based on the two bind variables, and then it will bind the result outwards to a location in the Data DOM as XML.

The first and second fm:using statements show the different syntax that can be used. For the first fm:using statement, only one attribute is set and the XPath is specified as a text node, however for the second statement, everything is specified, overriding FOX's own guess.

A *fm:api* using a DML statement might look like this:

```
  <fm:api name="api name">
    <fm:statement>
UPDATE trainingmgr.department d
SET d.location = 'London'
WHERE d.id = :dept_id
    </fm:statement>
    <fm:using name=":dept_id">ID</fm:using>
  </fm:api>
```

**NB: DML statements are not auto committed by the *fm:api*, you need to manually commit any changes using**

```
  <fm:transaction operation="COMMIT"/>
```

*It should also be noted that the value of 'operation' is case-sensitive.*

## Notes

Never issue a manual PL/SQL COMMIT statement inside an fm:api command, and remember to check that any PL/SQL functions or procedures referenced do not contain any COMMIT statements. All of the code run inside an fm:api should be transaction-safe, so that if a subsequent error occurs the entire workload can be rolled back. This way, the database state is preserved; either everything succeeds, or nothing is changed.

## Related

- fm:query
- fm:db-interface
- fm:run-api
- fm:using
- fm:statement

# FOX:Module:application-title

## Description

***fm:application-title*** is part of ***fm:header*** and allows for setting out text which describes application of which your module is part of. The text entered is populated as part of the ***:{sys}*** DOM under the heading ***application-title***. This in turn is extracted by the ***LAYOUT1LIB*** library module, added to a buffer and included as part of your module's html markup.

## Syntax

```
<fm:header>
  <fm:application-title>(Application title text)</fm:application-title>
</fm:header>
```

## Examples

Given this as the state of the module.

```
  <fm:header>
    <fm:application-title>My Application</fm:application-title>
    ...
  </fm:header>
```

This would be the result in ***:{sys}*** when accessing that module.

```
<module>
  <application-title>My Application</application-title>
  ...
</module>
...
```

This in turn would show on screen as a header with the text contents of ***fm:applicaton-title*** through ***'LAYOUT1LIB***"s formatting

Which is effectively doing this:

```
<fm:set-buffer name="buffer-title1">
  <fm:expr-out match=":{sys}/module/application-title/text()"/>
</fm:set-buffer>
```

This buffer is then included in an appropriate part of the page.

### Notes

***LAYOUT1LIB*** Is a formatting library, therefore may be subject to change (or not used at all in your module).

### Related

- fm:header

# FOX:Module:attr

## Description

***fm:attr*** is a child element of ***fm:display-attr-list***. This element targets markup/display attributes on elements & set-outs and assigns a default value to be used. The default cascades through the libaried in modules that contribute to the presentation buffers. (i.e. if I set the default of ***prompt***, all elements capable of using ***prompt*** will be set to the value provided by ***fm:attr***.

Display attributes have an overriding hierarchy similar is functionality to CSS attributes and follows the list below:

- ***xs:element*** schema level is highest
- ***fm:set-out/fm:menu-out*** and associated markup is overridden by schema level
- ***fm:attr*** is overridden by ***fm:set-out*** (etc)

## Syntax

```
<fm:attr name="[ Name of attribute ]>[ State of attribute ]</fm:attr>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | Valid display attribute | The name of a display attribute of which to set the default safe of. | Yes |

## Examples

```
<fm:attr name="prompt">ChangeMe</fm:attr>
```

The above code will act as a default for any elements which do not have their ***prompt*** attribute set when being displayed.

## Related

- fm:display-attr-list

# FOX:Module:authentication

## Description

***fm:authentication*** is a child element of ***fm:control*** and controls whether or not to allow users onto the module. Given the ***fm:authentication*** state of ***not-required***, any user is allowed onto a module. However, given ***required*** for a user to access the module they would need to be logged in and hold the correct permissions for access.

## Syntax

```
<fm:control>
  <fm:authentication>[ required | not-required ]</fm:authentication>
</fm:control>
```

## Examples

Users are required to be logged in with the correct permissions for access.

```
  <fm:authentication>required</fm:authentication>
```

Users can access the module freely without need to log in.

```
  <fm:authentication>not-required</fm:authentication>
```

## Notes

Generally most modules are in the ***fm:authentication*** state of ***required*** as doing so allows for more control over what a user can and cannot do on the system. Exceptions to this would be login/registration modules and help screens.

## Related

- fm:control

# FOX:Module:build-notes

## Description

***fm:build-notes*** is a child of ***fm:header***. It is used to store notes related to release/testing the module.

## Syntax

```
<fm:header>
  <fm:build-notes>[ Build Notes ]</fm:build-notes>
  ...
</fm:header>
```

## Examples

```
<fm:build-notes>Must have datapatch0001.sql applied before testing</fm:build-notes>
```

## Notes

Deprecated, typicaly this information is now handled by an external release tool.

## Related

- fm:header

# FOX:Module:cache-key

## Description

***fm:cache-key*** is an internal memory locator used as part of ***fm:storage-location***. This provides the fox engine with a unique key in which to cache the contents of ***:{root}*** for faster updates to the database. Typically the cache-key is unique and generated upon each call of the module (achieved through using-types), however there are certain scenarios where by you may want to read from a cached storage-location from multiple instances of a module (mapsets), in this case a non-unique key can be used to allow multiple instances access to the mapset cache.

## Syntax

```
<fm:cache-key string="module name :bind">
  <fm:using [ using-type="(using type)" ]>bind xpath</fm:using>
</fm:cache-key>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| string | xs:string | Key used to identify storage location cache, can use numbered bind such as ***:1*** corresponding to the first ***fm:using*** bind type. | Yes |

## Examples

Typical unique cache key.

```
<fm:cache-key string="MY_MODULE:1">
  <fm:using using-type="UNIQUE"/>
</fm:cache-key>
```

Typical non-unique cache key

```
<fm:cache-key string="MY_MODULE:1">
  <fm:using>:{params}/ID</USING>
</fm:cache-key>
```

## Notes

Only a finite amount of storage locations can be in memory at any one time (around 50-200 depending on the instance of FOX engine that's being used) each of these has a time to live (TTL) from last use (around 15-60 minutes again depending on environment). Once the TTL expires the storage location get's removed from memory and recreated when necessary, this can be forced via the use of bang command ***!PURGE***

## Related

- fm:storage-location
- fm:using

# FOX:Module:comments

## Description

***fm:comments*** is a child element of ***fm:documentation*** and is used for storing text comments relevant to developers.

## Syntax

```
<fm:comments>[ Comment ]</fm:comments>
```

## Examples

```
<fm:comments>Only used there is list in param DOM.</fm:comments>
```

## Related

- fm:documentaion

# FOX:Module:control

## Description

***fm:control*** is a container element for FOX commands related to controlling module behavior in the FOX engine.

***fm:authentication*** is the only section of ***fm:control*** that has much use in the current version of the fox engine. All other commands are here for syntactical reference and are ignored by the FOX engine. Additional commands may be added to this section in future releases of FOX.

- ***fm:authentication*** Defines whether authentication is needed for this module.
- ***fm:transaction-mode*** Deprecated, FOX Engine will ignore.
- ***fm:transaction-mode-new*** Deprecated, FOX Engine will ignore.
- ***fm:transaction-commit*** Deprecated, FOX Engine will ignore.
- ***fm:transaction-procedural-state*** Deprecated, FOX Engine will ignore.
- ***fm:xml-commit*** Deprecated, FOX Engine will ignore.

## Syntax

```
<fm:module>
  <fm:control>
    <fm:authentication>[ Authentication Mode ]</fm:authentication>
    <fm:transaction-mode>[ Transaction Mode ]</fm:transaction-mode>
    <fm:transaction-mode-new>[ Transaction Mode ]</fm:transaction-mode-new>
    <fm:transaction-commit>[ Transaction Commit Mode ]</fm:transaction-commit>
    <fm:transaction-procedural-state>[ Transaction Procedural State Mode ]</fm:transaction-procedural-state>
    <fm:xml-commit>[ XML Commit Mode ]</fm:xml-commit>
  </fm:control>
  ...
</fm:module>
```

## Notes

The deprecated commands listed here will cause no effect on any FOX module.

## Related

- fm:authentication
- fm:transaction-mode
- fm:transaction-mode-new
- fm:transaction-commit
- fm:transaction-procedural-state
- fm:xml-commit

# FOX:Module:css

## Description

***fm:css*** is a child element of ***fm:css-list*** and provides functionality to link CSS into the html which FOX generates. The style sheets need to be defined as a record in ***envmgr.fox_components*** or the relative app-mnemonics component table, with a type of ***text/css***. The text specified as the contents of ***fm:css*** is linked directly to the name of a text/css type component from the modules component table.

## Syntax

```
<fm:css>[ Name of CSS from FOX Components table ]</fm:css>
```

## Examples

```
<fm:css>css/myStyleSheet</fm:css>
```

The above example shows a valid style sheet include into the module, below show that same style sheet being recalled from the database.

```
SELECT
    name
  , type
FROM envmgr.fox_components
WHERE type = 'text/css'
  AND name = 'css/myStyleSheet'
```

More often that not, ***fm:css*** should only be used for library modules that are specifying the style for your whole application area. An example of this would to create a wrapper for ***LAYOUT1*** specify any custom css code, then to library in the wrapper instead of ***LAYOUT1***. This is a commonly used design practice employed in most application areas.

***fm:css*** is only processed for the module at the ***TOP*** of the ***mod merge***, meaning that if you specify a ***fm:css-list*** tag, you must specify all CSS locations that you want your module (and those which library in your module) to use. Additionally, to override CSS already listed in the ***fox_components*** table, creating and using a different app-mnemonic and storing the new style sheet there would be a better solution to copying in contents of the old CSS file under a different location.

## Related

- fm:css-list
- fm:presentation
- fm:library

# FOX:Module:css-list

## Description

***fm:css-list*** is a module level container for ***fm:css*** elements.

## Syntax

```
<fm:css-list>
  <fm:css>...</fm:css>
</fm:css-list>
```

## Related

- fm:module
- fm:css

# FOX:Module:current-of-item

## Description

***fm:current-of-item*** is a child element of ***fm:row-lock*** this feature has been deprecated and is now ignored by the FOX engine.

## Related

- fm:row-lock

# FOX:Module:data-mapping

## Description

***fm:data-mapping*** is a deprecated feature of ***fm:parameter***

## Syntax

```
<fm:data-mapping>...</fm:data-mapping>
```

## Related

- fm:parameter

# FOX:Module:data-type

## Description

***fm:data-type*** is a deprecated feature of ***fm:parameter***

## Syntax

```
<fm:data-type>...</fm:data-type>
```

## Related

- fm:parameter

# FOX:Module:database

## Description

***fm:database*** is a database connector for storage locations which allows for the reading/writing/creating of a database record containing the ***:{root}*** DOM through PL/SQL, similar in functionality to ***fm:api*** and ***fm:query***.

***fm:query*** DML for selecting & locking row containing XMLType/LOB data for use as initialising ***:{root}*** to.

***fm:lock*** Deprecated/Ignored element, lock row using FOR UPDATE in select statement.

***fm:insert*** DML for creating a record should ***fm:query*** not return a usable result.

***fm:update*** DML triggered upon updating contents of the ***:{root}*** DOM. Does not need to include a bind to update LOB column as this is handles automatically via the LOB Locator.

***fm:delete*** Markup specific to ***fm:file-storage-location***, describes a DML statement to be executed upon the removal of the widget which links to this specific ***fm:file-storage-location***

## Syntax

```
<fm:database>
  <fm:query>
    <fm:sql>
SQL statement with :bind
    </fm:sql>
    <fm:using>bind xpath</fm:using>
  </fm:query>
  <fm:lock>
Deprecated
  </fm:lock>
  <fm:insert>
    <fm:sql>
DML statement with :bind
    </fm:sql>
    <fm:using>bind xpath</fm:using>
  </fm:insert>
  <fm:update>
    <fm:sql>
DML statement with :bind
    </fm:sql>
    <fm:using>bind xpath</fm:using>
  </fm:update>
</fm:database>
```

***fm:file-storage-location*** has a specific markup element set apart from the shared markup between ***fm:storage-location***

```
<fm:database>
  <fm:delete>
    <fm:sql>
[ DML statement with :bind ]
```

```
    </fm:sql>
  </fm:delete>
</fm:database>
```

## Examples

Majority of information is the following examples are snippets from fm:storage-location

```
<fm:database>
  <fm:query>
    <fm:sql>
SELECT xml_data FROM portal_folders WHERE id = :1
FOR UPDATE OF xml_data NOWAIT
    </fm:sql>
    <fm:using>:{params}/P_PF_ID</fm:using>
  </fm:query>
  <fm:insert>
    <fm:sql>
INSERT INTO portal_folders(
  id
  xml_data)
VALUES(
  :1
  :2)
    </fm:sql>
    <fm:using>:{params}/P_PF_ID</fm:using>
    <fm:using using-type="DATA-XMLTYPE"/>
  </fm:insert>
  <fm:update>
    <fm:sql>
UPDATE portal_folders SET id = id WHERE id = :1
    </fm:sql>
    <fm:using>:{params}/P_PF_ID</fm:using>
  </fm:update>
</fm:database>
```

***fm:query*** Attempts to select back a single row containing an XMLType which to initialise ***:{root}*** to.

Failing that, ***:{root}*** is initialized to the values discussed as part of ***fm:new-document***, after which ***fm:insert*** is run.

***fm:insert*** will insert a row into the database with the using binds defined, for ***:{root}*** to be stored you must bind the using-type "DATA-XMLTYPE".

***fm:update*** is run once per page churn whenever changes to ***:{root}*** arise, this is used mainly for kicking off database triggers which will not work on LOB based data changes (such as for XVIEWs)

In addition to the shared syntax shown above, ***fm:delete*** is used exclusively as part of ***fm:file-storage-location*** for removing file records from tables.

```
<fm:database>
  <fm:delete>
    <fm:sql>
DELETE FROM table_name
```

```
WHERE id = :1
    </fm:sql>
    <fm:using>:{theme}/WIDGET_NAME/file-id</fm:using>
  </fm:delete>
</fm:database>
```

### Related

- fm:storage-location
- fm:file-storage-location
- fm:query
- fm:lock
- fm:update
- fm:insert
- fm:delete
- fm:sql

# FOX:Module:db-interface

## Description

***fm:db-interface*** provides SQL and PL/SQL functionality from within a fox module, including features designed for integrating XMLType data into a DOM. This element contains 2 key components of a FOX module functionality

- Ability to Insert/Update/Delete data from database in a transaction safe way and Selecting results back into the FOX module.
- Ability to call stored functions and procedures, especially important for any workflow (Buisness Process) related functionality (Almost every FOX Project has made use of this).

These features are achieved by the following FOX markup.

- ***fm:table*** Defines processing rules for table updates, inserts and deletes, this is rarely used - kept in for legacy reasons.
- ***fm:query*** Returns results of an SQL query into a specified DOM.
- ***fm:api*** Anonymous PL/SQL block OR DML statement, used for any database interaction that is not purely querying out data.

## Syntax

```
<fm:db-interface>
  <fm:table>[ TABLE Locking ]</fm:table>
  <fm:query>[ SELECT Statenent ]</fm:query>
  <fm:api>PL/SQL or DML Statement</fm:api>
</fm:db-interface>
```

## Examples

```
<fm:db-interface name="dbint-product">
  <fm:loc
  <fm:query name="qry-product">
    <fm:select>
SELECT
    product_name
  , product_desc
  , product_price
FROM productgmr.products
WHERE product_id = :product_id
    </fm:select>
    <fm:using name=":product_id">:{params}/PRODUCT_ID/text()</fm:using>
  </fm:query>
  <fm:api name="api-update-description">
    <fm:statement>
BEGIN
  UPDATE productmgr.produtcs
  SET product_desc = :product_desc
  WHERE product_id = :product_id
END;
    </fm:statement>
    <fm:using name=":product_id">:{params}/PRODUCT_ID/text()</fm:using>
    <fm:using name=":product_desc">:{theme}/PRODUCT/PRODUCT_DESC/text()</fm:using>
  </fm:api>
</fm:db-interface>
```

## Notes

***fm:db-interface*** defines database interactions but will not run them, to cause a query or statement to be processed a Command has to be used inside a ***fm:do*** block. Each child of ***fm:db-interface*** has it's own Command, these are ***fm:run-api***, ***fm:run-query*** and ***fm:run-dml***, you can find out more about these Commands from the child element each is related too.

## Related

- fm:db-interface-list
- fm:table
- fm:query
- fm:api

# FOX:Module:db-interface-list

## Description

***fm:db-interface-list*** is child of ***fm:module*** and a container for repeating ***fm:db-interface*** elements.

## Syntax

```
<fm:module>
  <fm:db-interface-list>
    <fm:db-interface/>
    ...
  </fm:db-interface-list>
</fm:module>
```

## Related

- fm:module
- fm:db-interface

# FOX:Module:delete

## Description

***fm:delete*** is a child element of ***fm:database*** only valid when also a decendant of ***fm:file-storage-location***. It contains DML which will trigger once the XML node the file-storage-location is using has been removed. Removing a file upload node will trigger ***fm:delete***, allowing the file associated to said node before removed from the database if required.

Generally considered bad practice to use this command, keeping the uploads is preferred.

## Syntax

```
<fm:delete>
  <fm:sql>
[ DELETE DML ]
  </fm:sql>
  <fm:using>WIDGET_XPATH</fm:using>
</fm:delete>
```

## Examples

```
<fm:delete>
    <fm:sql>
DELETE FROM table_name
WHERE id = :1
    </fm:sql>
    <fm:using>:{theme}/WIDGET_NAME/file-id</fm:using>
  </fm:delete>
```

## Related

- fm:database
- fm:file-storage-location

# FOX:Module:description

## Description

***fm:description*** is a child of ***fm:header***. It is used to store a text description of the module, used by application developers to identify the modules purpose.

## Syntax

```
<fm:header>
  <fm:description>[ Description of module ]</fm:description>
  ...
</fm:header>
```

## Examples

```
<fm:description>Provide the searching functionality of the Product Search Application</fm:description>
```

## Related

- fm:header

# FOX:Module:display-attr-list

## Description

***fm:display-attr-list*** is a child element of ***fm:presentation*** and contains repeating ***fm:attr*** elements. These provide defaults for unset attributes which cascade throughout all modules libraried into the current module.

## Syntax

```
<fm:display-attr-list>
  <fm:attr name="[ Attribute name ]">[ Atribute State ]</fm:attr>
  ...
</fm:display-attr-list>
```

## Related

- fm:presentation
- fm:attr

# FOX:Module:documentation

## Description

***fm:documentation*** contain useful developer information about the parent element under which this is located.

## Syntax

```
<fm:documentation>
  <fm:comments>[ Comment ]</fm:comments>
  <fm:description>[ Description ]</fm:description>
  <fm:pre-condition>[ Pre-Condition ]</fm:pre-condition>
  <fm:parameter-list>*[ Paramters ...]</fm:parameter-list>
  <fm:name-space-list>*[ Name spaces]<fm:name-space-list>
</fm:documentation>
```

## Examples

```
<fm:documentation>
   <fm:comments>This module is using data from a legacy application</fm:comments>
   <fm:description>Product Searching</fm:description>
   <fm:pre-condition>Parameter IN_LIST must contain an element to search from</fm:pre-condition>
</fm:documentation>
```

### Notes

Generally unused, other elements and comments typically convey information well enough.

- ***fm:parameter-list*** is only applicable when ***fm:documentation*** is a descendant of ***fm:entry-theme***
- ***fm:name-space-list*** is only applicable when ***fm:documentation*** is a descendant of ***fm:header***

### Related

- fm:header
- fm:action
- fm:comments
- fm:description
- fm:pre-condition
- fm:parameter-list
- fm:name-space-list

# FOX:Module:entry-theme

## fm:entry-theme

### Description

```
<fm:entry-theme name="String" [ authentication-type="..." published="..." type="..." ]>
  <fm:storage-location>...</fm:storage-location>
  <fm:state>...</fm:state>
  <fm:attach>XPath</fm:attach>
  <fm:do>
    [command list]
  </fm:do>
</fm:entry-theme>
```

Defines an entry-theme, which acts as an entry point into the module.

- **<fm:storage-location>** - The Data DOM Storage Location
- **<fm:state>** - The initial module state when the entry theme is used
- **<fm:attach>** - The initial attach point. The XPath expression is run relative to the <*fm:root-element*> in the specified storage-location
- **<fm:do>** - List of FOX commands run when the entry theme is used.

### Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String Literal | Specifies the name of the entry-theme. This will be used when calling modules with fm:call-module | Yes |
| type | String Literal | • internal - calling the entry theme directly (via url) is not allowed<br>• external - can call the entry theme directly (via url)<br>• service-rpc - used for web services<br>• service-document - used for web services | No |
| authentication-type | String Literal | • portal - uses the cookie set by logging into the portal to authenticate the user<br>• http - uses http authentication to authenticate the user | No |

## Notes

See fm:param for detailed instruction on FOX Web services and implementation by ***fm:entry-theme***

## Related

- fm:entry-theme-list
- fm:storage-location
- fm:state
- fm:attach
- fm:do

# FOX:Module:entry-theme-list

## Description

Container element for multiple ***fm:entry-theme*** elements.

## Syntax

```
<fm:module>
...
  <fm:entry-theme-list>
    <fm:entry-theme name="new">
    ...
    <fm:entry-theme>
  </fm:entry-theme-list>
...
</fm:module>
```

## Related

- fm:module
- fm:entry-theme

# FOX:Module:file-storage-location

## Description

***fm:file-storage-location*** is an extension of functionality defined by ***fm:storage-location*** (though internally the code is likely to differ greatly), most of the module markup elements behave identically to storage-location. The major difference between the two is that ***fm:file-storage-locations*** works in conjunction with schema element attributes such as ***widget="file"*** allowing users to upload files and have them stored in a table as a BLOB as opposed to storing XML data for a module to use.

The Upload Widget is used to upload files to the database. It is linked to a file-storage-location which allows FOX to control adding, removing and updating files in the database. Each uploaded file is assigned a unique file-id which can be used to reference it.

## Syntax

Module markup

```
  <fm:file-storage-location name="[Storage Location Name]">
    <fm:cache-key string="module name + other text"/>
    <fm:database>
      <fm:query>
        <fm:sql>
[ SELECT ]
        </fm:sql>
      </fm:query>
      <fm:insert>
        <fm:sql>
[ INSERT ]
        </fm:sql>
      </fm:insert>
      <fm:update>
        <fm:sql>
[ UPDATE ]
        </fm:sql>
      </fm:update>
      <fm:delete>
        <fm:sql>
[ DELETE ]
        </fm:sql>
      </fm:delete>
    </fm:database>
  </fm:file-storage-location>
```

Schema element markup

```
<xs:element name="string" type="file-type" ns:edit="." fox:file-storage-location="string" fox:widget="file"
             fox:upload-mode="no-interrupt-modal-upload-window"
fox:upload-window-type="modal"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | xs:string | Name of the storage location, this should have a prefix of 'sl' to denote storage location (i.e sl-search-results). | Yes |

## Schema Widget Syntax

- **type="file-type"** will mark this element as being used for file uploads.
- **ns:widget="file"** will allow you to upload new files or view and download previously uploaded files.
- **ns:file-storage-location="storage-location-name"** is required to specify the file storage location that is to be used by your upload widget.
- **ns:edit="XPath"** attribute must evaluate to true if you want to be able to upload new files. By not including this attribute or providing an XPath that evaulates to false you can restrict your upload widget to be used for viewing or downloading previously uploaded files.
- **ns:upload-mode** can be one of three values:
  - **interrupt-on-module-push-pop-upload-target-any** - Causes the upload to be interrupted on a module push/pop. If your upload is to a transient DOM (i.e. theme) then a push or pop will cause the upload target DOM element to be inaccessible, so the upload must be cancelled.
  - **no-interrupt-modal-upload-window** - Specify this when the window type is modal - in this case, it is not possible to transform the callstack during an upload.
  - **no-interrupt-upload-target-storage-location** - Specify this when the upload is to the :{root} DOM, and is modeless. The root DOM is still accessible to FOX even after a callstack transformation. Uploads in this mode will not be interrupted, allowing the user to continue to work unimpeded in the parent window, without having to wait for their upload to finish.
- **ns:upload-window-type** is used to define the upload window as modal or modeless
- **ns:upload-widget-style** can be used to control the display of the widget

## Examples

```
<!-- Example of storage location -->
<fm:file-storage-location name="sl-files">
  <fm:cache-key string="DOC_UPLOADER :1">
    <fm:using using-type="UNIQUE"/>
  </fm:cache-key>
  <fm:database>
    <fm:query>
      <fm:sql>
SELECT file_content
FROM document_storage
WHERE file_id = :1
```

```
FOR UPDATE OF file_content
    </fm:sql>
      <fm:using>DOCUMENT_UPLOAD_WIDGET/file-id</fm:using>
    </fm:query>
    <fm:insert>
      <fm:sql>
INSERT INTO document_storage(
    file_id
  , file_description
  , file_content
  , created_date
  , created_by)
VALUES(
    :1
  , :2
  , empty_blob()
  , SYSDATE
  , :3)
    </fm:sql>
      <fm:using>DOCUMENT_UPLOAD_WIDGET/file-id</fm:using>
      <fm:using>DOCUMENT_UPLOAD_WIDGET/captured-fields/description</fm:using>
      <fm:using>:{user}/WUA_ID</fm:using>
    </fm:insert>
    <fm:update>
      <fm:sql>
UPDATE document_storage SET file_id = file_id
WHERE file_id = :1
    </fm:sql>
      <fm:using>DOCUMENT_UPLOAD_WIDGET/file-id</fm:using>
    </fm:update>
    <fm:delete>
      <fm:sql>
DELETE FROM document_storage
WHERE file_id = :1
    </fm:sql>
      <fm:using>DOCUMENT_UPLOAD_WIDGET/file-id</fm:using>
    </fm:delete>
  </fm:database>
</fm:file-storage-location>


<!-- Example of schema element as wiget -->
<xs:element name="DOCUMENT_UPLOAD_WIDGET" type="file-type" ns:edit="." fox:file-storage-location="sl-files" fox:widget="file"
             fox:upload-mode="no-interrupt-modal-upload-window"
fox:upload-window-type="modal"/>
```

The above code creates an upload widget named ***DOCUMENT_UPLOAD_WIDGET*** which will save all uploaded files to the table ***document_storage***. When a file is uploaded, the file_content field of the table will be populated with the BLOB val of the uploaded file. Upon the removal of the upload widget from the schema the record

containing the file-id will be removed from the table.

Below is a typical file-storage table needed by FOX to successfully save files to.

```
CREATE TABLE document_storage (
  file_id             VARCHAR2(30)     NOT NULL
, file_description    VARCHAR2(4000)
, file_content        BLOB             NOT NULL
, created_date        DATE
, created_by          NUMBER(20)
)
TABLESPACE tbsdata;

GRANT INSERT, UPDATE, DELETE, SELECT ON document_storage TO appenv;
```

## Related

- fm:storage-location-list
- fm:storage-location
- fm:cache-key
- fm:database

# FOX:Module:for-each-fetch

## Description

***fm:for-each-fetch*** is a child element of ***fm:query*** and is used to process individual records as they are returned from the database into the DOM. Per record, the ***fm:do*** block of for-each-fetch is run, with it's context set to the latest record returned. This allows for personalized processing of records before they are stored into the DOM, and without the need for a for-each command to be run separately after the query.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| post-dom-change | Y/N | After database read and ***after*** XML DOM population of (Y), Default is Y | No |
| pre-dom-change | Y/N | After database read and ***before*** XML DOM population of (Y), Default is N | No |

## Examples

```
<fm:query name="qry-product-price">
  <fm:target-path match="PRODUCT"/>
  <fm:select>
SELECT
    id
  , price
FROM productmgr.products
  </fm:select>
  <fm:for-each-fetch post-dom-change="Y">
```

```
    <fm:do>
      <fm:assign initTarget="./PRICE_DOLLAR" expr="number(./PRICE) * number(1.5)"/>
    </fm:do>
  </fm:for-each-fetch>
</fm:query>
```

The above example will return all products from the products table, before ***fm:for-each-fetch*** is run on the first record, the state of the DOM looks like this.

```
<PRODUCT>
  <ID>1</ID>
  <PRICE>12</PRICE>
</PRODUCT>
```

***fm:for-each-fetch*** is then invoked for that particular record and executing the ***fm:do*** block using data in the DOM (as described by the attribute ***post-dom-change***) which results in the following

```
<PRODUCT>
  <ID>1</ID>
  <PRICE>12</PRICE>
  <PRICE_DOLLAR>18.0</PRICE_DOLLAR>
</PRODUCT>
```

This is repeated for every record returned, akin to a cursor for-loop. Like the cursor for loop, for large result sets this will become increasingly slow and resource hogging.

## Related

- fm:query
- fm:do

# FOX:Module:header

## Description

***fm:header*** is a container for elements that store meta data about your fox module, this data is primarily used to describe your module to the database and other developers. However ***fm:title*** and ***fm:application-title*** are often shown as html headers to your module that end users would use to identify the page they are on.

- ***fm:name*** Identifier of your module to the database and fox engine. This must be unique and match the file name which your module is saved under.
- ***fm:title*** A short title which area of your application your module applies to. (i.e Search Screen)
- ***fm:application-title*** A short title which your entire application is known by. (i.e Product Management System)
- ***fm:documentation*** Contains child elements which provide useful developer information, such as parameters, namespace list and comments. Deprecated.
- ***fm:version-no*** Current version number of this module, this is usually linked to a version control system.
- ***fm:version-desc*** Description of the current version, usually linked to a version control system.
- ***fm:history*** Full list of changes since first committed to a version control system
- ***fm:description*** Basic text description of the module.
- ***fm:build-notes*** Notes related to environment (database/engine) specific prerequisites. Deprecated.
- ***fm:help-text*** Non essential notes that may be of interest to other application develope

## Syntax

```
<fm:module>
  <fm:header>
    <fm:name>[ MODULE_NAME ]</fm:name>
    <fm:title>[ Title ]</fm:title>
    <fm:application-title>[ Application title ]</fm:application-title>
    <fm:documentation>...</fm:documentation>
    <fm:version-no>[ Version Number ]</fm:version-no>
    <fm:version-desc>[ Version Description ]</fm:version-desc>
    <fm:history>[ History of changes ]</fm:history>
    <fm:description>[ Module Description ]</fm:description>
    <fm:build-notes>[ Build Notes ]</fm:build-notes>
    <fm:help-text>[ Help Text ]</fm:help-text>
  </fm:header>
  ...
</fm:module>
```

## Examples

A typical ***fm:header*** will look as follows.

```
  <fm:header>
    <fm:name>PRODUCT_SEARCH</fm:name>
    <fm:title>Product Search Area</fm:title>
    <fm:application-title>Product Management System</fm:application-title>
    <fm:documentation>...</fm:documentation>
    <fm:version-no>1.3</fm:version-no>
```

```
    <fm:version-desc>Updated search query to include new field.</fm:version-desc>
    <fm:history>
      1.3  Jane   Updated search query to include new field.
      1.2  Jane   Changed formatting of results.
      1.1  Tom    Added mapset to select product from.
      1.0  Tom    Initial Revision
    </fm:history>
    <fm:description>Search module for product information</fm:description>
    <fm:build-notes/>
    <fm:help-text>This module relies off data being fed in via PRODUCT_ADD</fm:help-text>
  </fm:header>
```

## Related

- fm:name
- fm:title
- fm:application-title
- fm:documentation
- fm:version-no
- fm:version-desc
- fm:history
- fm:description
- fm:build-notes
- fm:help-text

# FOX:Module:help-text

## Description

***fm:help-text*** is a child of ***fm:header***. It is used to store notes that may be useful to an application developer.

## Syntax

```
<fm:header>
  <fm:help-text>[ Help Text ]</fm:help-text>
  ...
</fm:header>
```

## Examples

```
<fm:help-text>A lot of buffers from LIBRARY001L have been overridden</fm:help-text>
```

## Related

- fm:header

# FOX:Module:history

## Description

***fm:history*** is a child of ***fm:header***. It is used to store a full version history from initial commit up to the latest version. Version control software should target this element to update upon every commit.

## Syntax

```
<fm:header>
  <fm:history>[ Version History ]</fm:history>
  ...
</fm:header>
```

## Examples

```
<fm:header>
  <fm:history>
     1.3  Jane   Updated search query to include new field.
     1.2  Jane   Changed formatting of results.
     1.1  Tom    Added mapset to select product from.
     1.0  Tom    Initial Revision
  </fm:history>
</fm:header>
```

## Related

- fm:header

# FOX:Module:insert

## Description

***fm:insert*** is run when ***fm:query*** returns no rows in order to create a record for subsequent storage location writes. Because the query returned no retults, ***:{root}*** is initialised to the element name described in ***fm:new-document***.

## Syntax

```
<fm:insert>
  <fm:sql>
[ SQL Update Statement ]
  </fm:sql>
  <fm:using using-type="DATA-XMLTYPE">:bind xpath</fm:using>
</fm:insert>
```

## Examples

***fm:insert*** works in the same fashion as any other DML INSERT statement, except when in comes to binding in the ***:{root}*** DOM. For your insert to work properly, using-type="DATA-XMLTYPE" must be defined for the xml data column of the table you wish to insert into. This contradicts how ***fm:update*** behaves, this is due to ***fm:update*** working on data already existing, and ***fm:insert*** creating the data in the first place.

```
<fm:insert>
  <fm:sql>
INSERT INTO portal_folders(
  id
  xml_data
)
VALUES(
  :1
  :2
)
    </fm:sql>
  <fm:using>:{params}/P_PF_ID</fm:using>
  <fm:using using-type="DATA-XMLTYPE"/>
</fm:insert>
```

## Note

Use of ***fm:insert*** is generally discouraged, the same affect can be accomplished through the use of ***fm:api*** in a more controlled manner.

## Related

- fm:database
- fm:using
- fm:sql

# FOX:Module:into

## Description

***fm:into*** is a child element ***fm:query*** and is used to specify the output location in a DOM of a specific column from the query.

## Syntax

```
<fm:into name="Column Name" datadom-type="xs:type" sql-type="sql:type" datadom-location="XPath location of DOM"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String identifier of column | Column which to location via this ***fm:into*** | Yes |
| datadom-type | xs:type | Type of data which the contents of the column will behave as<br>• dom<br>• xs:date<br>• xs:datetime<br>• xs:string<br>• xs:time | No |
| sql-type | SQL internal types | The internal SQL type of data contained by the column<br>• varchar<br>• date<br>• xmltype<br>• clob | No |
| datadom-location | XPath String | XPath location of output, relative to target-path/match | No |

## Examples

```
<fm:query name="qry-sysdate">
  <fm:select>
SELECT
    id
  , xml_data
FROM prodmgr.product_data
WHERE id = :product_id
```

```
  </fm:select>
  <fm:using name=":product_id">:{params}/PRODUCT_ID/text()</fm:using>
  <fm:into name="xml_data" datadom-location="./PRODUCT_DATA" datadom-type="dom" sql-type="xmltype"/>
</fm:query>
```

The above statement will output the contents of the query into whichever path it's been matched on in ***fm:run-query*** say for this example ***:{theme}/MY_RESULT***. This dom will look as follows.

```
<MY_RESULT>
  <ID>ID of product</ID>
  <PRODUCT_DATA>Contents of xml_data column</PRODUCT_DATA>
</MY_RESULT>
```

## Related

- fm:query
- fm:run-query

# FOX:Module:key

## Description

***fm:key*** is the child element of ***fm:primary*** and holds the column name of either a primary of foreign key in the table specified by ***fm:table***

## Syntax

```
<fm:key>COLUMN_NAME</fm:key>
```

## Examples

```
<fm:key>PRODUCT_ID</fm:key>
```

## Related

- fm:primary

# FOX:Module:library

## Description

<fm:library> elements allow other FOX modules to be incorporated into the specified current module. The result can be seen in the "ModMerger" DOM available from the developers menu.

FOX takes the current module and looks at *fm:library-list*. All sections from each *fm:library* are included, so long as they are not already defined in the current module. If definitions [e.g. actions, set-buffers etc] are already defined [i.e. same name] in the current module, they are not libraried in, unless they are marked up with *stub-overload="."* attribute, in this case they are called stubs, (not the same as previously mentioned stub data) the content is over-written by the libraried in version. Entry themes are not libraried in.

Any namespace FOX finds in the definitions is inspected to see if it already exists in the current module. If it exists, FOX renames the namespace for the libraried in module to make it unique. There are two exceptions to this rule: *fox* is a namespace and is always global. other namespaces can by defined as global, and these are not renamed: *xmlns:orders="*http://www.og.dti.gov/fox_global*"*. At the end of the process FOX looks to the next *fm:library* for more modules to include. The version of the processed module becomes the 'current module' and the algorithm is recursive. Therefore any <fm:library> elements inside an already libraried module are also libraried in.

Libraried in components are then referenced just like local components.

## Attribute Summary

*None*

## Examples

```
<fm:library-list>
  <fm:library>LAYOUT1</fm:library>
</fm:library-list>
```

LAYOUT1 is the most common module libraried in. It contains an extensive set of buffers and actions to provide a generic layout for most FOX applications.

## Related

- <fm:library-list>

# FOX:Module:library-list

## Description

<fm:library-list> is a container for <fm:library> elements and allows other FOX modules to be imported, making their contents available in the parent module.

## Attribute Summary

*None*

## Examples

```
<fm:library-list>
  <fm:library>Module_Name</fm:library>
</fm:library-list>
```

## Related

- <fm:library>

# FOX:Module:lock

## Description

***fm:lock*** is now deprecated, locking can be achieved using FOR UPDATE OF in ***fm:query***'s select statement.

## Related

- fm:database

# FOX:Module:map-path

## Description

***fm:map-path*** is a child element of ***fm:table***, this markup has been deprecated.

## Syntax

```
<fm:map-path match="XPath"/>
```

## Related

- fm:table

# FOX:Module:map-set

## Description

The <fm:map-set> element is used to generate enumerated lists, with each enumeration mapping to a data key.

A map-set behaves like an Enumeration list of values, but is highly dynamic in nature. While the user selects from the 'keys', the user input is stored in XML as the corresponding 'data' (see examples below). The data value is usually a simple type (eg. xs:string), but it can also be a complex collection to allow greater flexibility.

## Concepts

Before a map-set can be used we need to define where it is held, a <fm:storage-location> (SL). Map-set values can be cached in memory only, or stored in the database and retrieved via a query. The cache key defines where in memory the map-set is held. Note the cache key comprises a static name part, and a unique part using bind variable. Without the unique bind variable, the map-set data will never change, because the storage location will not be changed each time it is loaded. See <fm:cache-key> for more information. For a map set the SL's <fm:root-element> is always 'map-set-list'. The storage location will ensure that the XML root element is initialized and no more.

The <fm:do> block code defines how the remainder of the XML is constructed (the key -> data mappings). This can be done through a database query or using a template. (See examples below) An example of the format of the XML the do block should generate is:

```
<map-set>
  <rec>
    <key>Male</key>
    <data>M</data>
  </rec>
  <rec>
    <key>Female</key>
    <data>F</data>
  </rec>
</map-set>
```

The <fm:refresh-timeout-mins> element specifies how long the constructed data is good for. If underlying data changes frequently, consider 5, 3 or even 0 mins. Otherwise 30 mins+.

A special value of 999999999 mins instructs FOX to never refresh the map set - not even the first time.

Note that a refresh may occur earlier than that specified, if XML was flushed from memory to free space.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | Literal String | The name used to reference the map-set | Yes |
| refresh | String 'Y' or 'N' (Boolean) | If 'Y', a refresh will be performed at intervals specified by the map set refresh element settings. No indicates Fox will never refresh or initially populate the map set by executing the child DO command. This means its up to the developer to manually load the map set (possibly in the storage location initialization). | Defaults to 'Y' |
| stub-overload | XPath String | If the XPath evaluates to true, any libraried in map-set with the same name will overwrite this one | No |

## Examples

A map-set using a run-query to define the key-data mappings.

```
<fm:map-set-list>
  <fm:map-set name="ms-mapset-example">
    <fm:storage-location>sl-ms-example</fm:storage-location>
    <fm:do>
      <fm:run-query interface="dbint-mapsets" query="qry-mapset"/>
    </fm:do>
    <fm:refresh-timeout-mins>0</fm:refresh-timeout-mins>
    <fm:refresh-in-background>true</fm:refresh-in-background>
  </fm:map-set>
</fm:map-set-list>
```

The query

```
<fm:query name="qry-mapset">
  <fm:target-path match="map-set/rec"/>
  <fm:select>
SELECT
  INITCAP(name) "key"
, deptno "data"
FROM trainingmgr.department
  </fm:select>
</fm:query>
```

And the storage-location

```
<fm:storage-location name="sl-ms-example">
  <fm:cache-key string="example-ms"/>
  <fm:new-document>
    <fm:root-element>map-set-list</fm:root-element>
  </fm:new-document>
</fm:storage-location>
```

This map set can now be referenced in a schema element, creating a drop-down list with all the 'key' elements. The corresponding 'data' element will then be inserted into the 'departments' element in the DOM.

```
<xs:element name="departments" fox:map-set="ms-mapset-example" fox:widget="selector" fox:key-null="Select if required"/>
```

### Related

- <fm:refresh-timeout-mins>
- <fm:refresh-in-background>
- <fm:storage-location>
- <fm:do>
- <fm:query>
- <fm:map-set-list>

# FOX:Module:map-set-list

### Description

<fm:map-set-list> is a container for <fm:map-set> elements used to generate enumerated lists with each enumeration mapping to a data key.

### Attribute Summary

*None*

### Examples

```
<fm:map-set-list>
  <fm:map-set>
    <fm:storage-location/>
    <fm:do/>
    <fm:refresh-timeout-mins/>
    <fm:refresh-in-background/>
  </fm:map-set>
</fm:map-set-list>
```

### Related

- <fm:map-set>
- <fm:storage-location>

# FOX:Module:matrix-and

## Description

***fm:matrix-and*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-author

## Description

***fm:matrix-author*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-container-record-number

## Description

***fm:matrix-container-record-number*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-date-closed

## Description

***fm:matrix-date-closed*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-date-created

## Description

***fm:matrix-date-created*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-date-registered

## Description

***fm:matrix-date-registered*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-external-reference

## Description

***fm:matrix-external-reference*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-into

## Description

***fm:matrix-into*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-notes

## Description

***fm:matrix-notes*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-or

## Description

***fm:matrix-or*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-order-by

## Description

***fm:matrix-order-by*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-record-number

## Description

***fm:matrix-record-number*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-record-title

## Description

***fm:matrix-record-title*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-record-type

## Description

***fm:matrix-record-type*** is a child of ***fm:matrix-search***, see fm:matrix-search

## Related

- fm:matrix-search

# FOX:Module:matrix-search

## fm:matrix-search

```
 <fm:matrix-search results-target-path="Single Node Complex XPath"
mode="Literal String" results-set-count="Single Node Complex XPath"
[results-set-size="XPath Integer" results-set-offset="XPath Integer"]

   [<fm:matrix-order-by name="''[[FOX:Reference:XPath_String|XPath
String]]'" [direction="''[[FOX:Reference:XPath_String|XPath
String]]'"]/>...]
   [<fm:matrix-into name="''Literal String'"
target-path="''[[FOX:Reference:Simple XPath|Simple XPath]]'"/>...]

   [<matrix-author words=""/>...]
   [<matrix-external-reference words=""/>...]
   [<matrix-notes words=""/>...]
   [<matrix-record-title names=""/>...]

   [<fm:matrix-container-record-number number=""/>...]
   [<matrix-record-number number=""/>...]

   [<matrix-date-closed from-date="" to-date=""/>...]
   [<matrix-date-created from-date="" to-date=""/>...]
   [<matrix-date-registered from-date="" to-date=""/>...]
```

```
  [<matrix-record-type names=""/>...]

  [<fm:matrix-or>
    [matrix search criteria]
  </fm:matrix-or>...]
  [<fm:matrix-and>
    [matrix search criteria]
  </fm:matrix-and>...]

</fm:for-each>
```

## Parameters

### results-set-size

**Default:** 20

### results-set-offset

**Default:** 0

## matrix-order-by Parameters

### direction

Will be ordered ascending if this evaluates "ascending", otherwise it will be order descending.

**Default:** "descending"

## matrix-into Parameters

**Default:** <fm:matrix-into name="NUMBER" target-path="."/>

### name

Comma-separated list.

**Default:** "NUMBER"

### target-path

**Default:** "."

# FOX:Module:mode-rule

## Description

<fm:mode-rule> is used within a <fm:security-list> and can be used to ENABLE a namespace for **edit** and **read-only** set-out or DISABLE it. The namespace, privilege, state and theme attributes can all contain comma separated conditions.

This provides a way of overriding active namespaces for set-out/menu-out. It is a further final way of filtering the content of the screen under different conditions.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| namespace | Literal String | The namespace to enable/disable. Can be a comma separated list. | Yes |
| operation | Literal String | ENABLE or DISABLE the namespace. | Yes |
| privilege | Literal String | The privilege(s) the user must have to view/edit the namespace. Can be a comma separated list. | No |
| state | Literal String | The sate(s) the user must be in to view/edit the namespace. Can be a comma separated list. | No |
| theme | Literal String | The entry theme(s) the user must be in to view/edit the namespace. Can be a comma separated list. | No |
| xpath | XPath String | The XPath test run relative to the root element. If it evaluates to true, the rule is applied. | No |
| rule-ref | Literal String | The name of a security-rule whose conditions to implement. | No |

## Examples

Generic:

```
<fm:security-list>
 <fm:mode-rule namespace="namespace" operation="ENABLE | DISABLE" [privilege="user privilege" state="state" theme="entry-theme" xpath="XPath" rule-ref="rule-name"]/>
</fm:security-list>
```

A mode-rule which disables the 'department-edit' namespace when the user is in the 'review' or 'print' state:

```
<fm:security-list>
 <fm:mode-rule namespace="department-edit" operation="DISABLE" state="review"/>
</fm:security-list>
```

## Related

- <fm:view-rule>
- <fm:security-rule>
- <fm:security-rule>

# FOX:Module:module

## Description

***fm:module*** is the root element for all fox module markup, this exists as a child element of ***xs:appinfo***

## Syntax

Position in the schema

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <fm:module>
        ...
      </fm:module>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

Content of ***fm:module***

```
<fm:module>
  <fm:header/>
  <fm:control/>
  <fm:security-list/>
  <fm:library-list/>
  <fm:storage-location-list/>
  <fm:entry-theme-list/>
  <fm:action-list/>
  <fm:db-interface-list/>
  [ <fm:css-list/> ]
  <fm:presentation/>
  <fm:state-list/>
  <fm:template-list/>
  <fm:mapset-list/>
  [ <fm:pagination-definition-list/> ]
</fm:module>
```

## Related

- fm:header
- fm:control
- fm:security-list
- fm:library-list
- fm:storage-location-list
- fm:entry-theme-list
- fm:action-list
- fm:db-interface-list
- fm:css-list
- fm:presentation
- fm:state-list
- fm:template-list
- fm:map-set-list
- fm:pagination-definition-list

# FOX:Module:name

## Description

***fm:name*** Is the identifier that the FOX engine will use for all processing. This must match the saved filename and be unique to the ***envmgr.fox_components*** table and all other application mnemonic's.

## Syntax

```
<fm:header>
  <fm:name>[ MODULE_NAME ]</fm:name>
  ...
</fm:header>
```

## Examples

***fm:name*** must match the filename minus it's extension. i.e If my filename is as follows:

```
MY_MODULE.xml
```

Then ***fm:name*** should look like this.

```
<fm:name>MY_MODULE</fm:name>
```

## Notes

It is considered good practice to use **uppercase** on all module names, as the FOX engine considers all names case sensitive. i.e MODULE_1 and module_1 are different modules to the engine.

## Related

- fm:header

# FOX:Module:name-space

## Description

***fm:name-space*** is a child element of ***fm:name-space-list*** and contains documentation about namespaces used inside the fox module that it is specified in.

## Syntax

```
<fm:name-space>
  <fm:name>[Name of namespace(s) being documented]</fm:name>
  <fm:description>[Description of namespace]</fm:description>
  <fm:comments>[Comments on use of namespace]</fm:comments>
</fm:name-space>
```

## Examples

```
<fm:name-space>
  <fm:name>email-list-add-user</fm:name>
  <fm:description>When adding users to the mailing list, use this namespace to access
associate functionality </fm:description>
  <fm:comments>You can remove a user using the default mail name space email-list</fm:comments>
</fm:name-space>
```

## Related

- fm:name-space-list

# FOX:Module:name-space-list

## Description

***fm:name-space-list*** is a child element of ***fm:documentation*** and a container element for the repeating element ***fm:name-space***. This contains information about various name spaces used in the fox module.

## Syntax

```
<fm:name-space-list>
  <fm:name-space>...</fm:name-space>
  ...
</fm:name-space-list>
```

## Related

- fm:documentation
- fm:header
- fm:name-space

# FOX:Module:new-document

## Description

***fm:new-document*** initializes the ***:{root}*** DOM to the child node ***fm:root-element*** when either no ***fm:database*** element is present or ***fm:query*** return no rows.

## Syntax

```
<fm:new-document>
  <fm:root-element>(Root Element Name)</fm:root-element>
</fm:new-document>
```

## Examples

```
<fm:new-document>
  <fm:root-element>FOLDER</fm:root-element>
</fm:new-document>
```

Will initialise ***:{root}*** to

```
<FOLDER/>
```

Such that the XPath :{root} will return the FOLDER element as the root element.

## Related

- fm:storage-location
- fm:file-storage-location
- fm:root-element

# FOX:Module:page-size

## Description

***fm:page-size*** is a child element of ***fm:pagination-definition*** and is part of FOX enhancements around ***fm:run-query2***. It specifies the result set size maximum per page.

## Syntax

```
<fm:page-size>[ Maximum page size ]</fm:page-size>
```

## Examples

```
<fm:page-size>20</fm:page-size>
```

In the above example, the page size of the pagination buffer will be set to 20 records, the query will fill up pages to their maximum before initializing a new page/chunk in the pagination cache.

## Related

- fm:pagination-definition
- FOX Pagination

# FOX:Module:pagination-definition

## Description

***fm:pagination-definition*** is a child element of ***fm:pagination-definition-list*** and part of the enhancements brought in by ***fm:run-query2***. This describes size of the pagination pull per page, and any processing involved either before or after the dataset has been pulled into the page. The built in FOX pagination, to summarise, stores the result of a query in a pagination cache on the database, this is then split into chunks relating to the size defined in ***fm:page-size*** and the current page being selected. The following chunks are then pulled down/loaded in the DOM upon request.

## Syntax

```
<fm:pagination-definition name="[ Page Definition Name ]">
  <fm:page-size>[ Page Size ]</fm:page-size>
  <fm:pre-page>
    <fm:do>...</fm:do>
  </fm:pre-page>
  <fm:post-page>
    <fm:do>...</fm:do>
  </fm:post-page>
</fm:pagination-definition>
```

## Examples

```
<fm:pagination-definition name="pd-results">
  <fm:page-size>50</fm:page-size>
  <fm:pre-page>
    <fm:do>
      <fm:remove match=":{root}/SEARCH_RESULTS/PRODUCTS[SELECT/SELECTED != 'true']"/>
    </fm:do>
  </fm:pre-page>
  <fm:post-page>
    <fm:do>
      <fm:call action="action-get-size-of-results"/>
    </fm:do>
  </fm:post-page>
</fm:pagination-definition>
```

The above example is a pagination definition for a products search screen which allows results to be carried over from page to page, this is achieved by removing the non-selected results from ***SEARCH_RESULTS*** before the new results are populated. The 'Result Count' typically displayed at the end of query is updated with the new size of the result set, this has to happen after pagination has returned the results.

***fm:pre-page*** if called, stops the FOX Pagination engine from removing the previous results as it is assumed you want to manually handle how to removed the old result set (as shown here in the example). ***fm:post-page*** does not share this functionality.

## Notes

***fm:pagination-definition*** only describes the layout and before/after processing of the query, it does not state where the results will go inside the DOM or when to fetch them. That is handled by ***fm:query*** and ***fm:run-query2***.

More information on the FOX Engine implementation of pagination can be found here FOX Pagination

## Related

- fm:pagination-definition-list
- fm:page-size
- fm:pre-page
- fm:post-page
- fm:query
- fm:run-query2
- FOX Pagination

# FOX:Module:pagination-definition-list

## Description

***fm:pagination-definition-list*** is an optional module level container for repeating ***fm:pagination-definition*** elements, this is part of a feature still in development relating to ***fm:run-query2*** and cached pagination of pages.

## Syntax

```
<fm:pagination-definition-list>
  <fm:pagination-definition name="[ Pagination Definition Name ]">...</fm:pagination-definition>
</fm:pagination-definition-list>
```

## Notes

Feature still in development, behavior may change unannounced as a result of this.

## Related

- fm:module
- fm:pagination-definition

# FOX:Module:param

## Description

***fm:param*** is child element of ***fm:param-list*** and part of the web service processing in the FOX Engine using SOAP. FOX accepts data in the specified SOAP wrapper and validates input data defined in ***fm:param***, processes the data using commands specified in the module, then returns the data to the calling user (or application) in the format as defined by ***fm:return***.

## Syntax

```
<fm:param name="Element Name" type="Element Content Type" xpath="Xpath" mand="Xpath" description="Text Description"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | XML Element Name | Name of XML element without namespace that is to be expected as input. | Yes |
| xpath | XPath location | Location of xml in DOM to validate input against | No |
| type | xs:<data-type> for leaf nodes, Element Name (LIST_XYZ) for branch nodes | Datatype to validate against, element name can be used as data-type | Yes |
| mand | XPath boolean | Describes if input XML element is mandatory | No |
| description | xs:string | Text description of parameter | No |

## Examples

```
Below is a basic example of how a FOX module behaves as a web service.
<fm:entry-theme name="GetMyTest" type="service-rpc">
  <fm:storage-location>sl-main</fm:storage-location>
  <fm:state>state-inital</fm:state>
  <fm:attach>/*</fm:attach>
  <fm:do>
    <fm:assign initTarget=":{return}/MY_OUTPUT" expr=":{params}/MY_DATA/text()"/>
  </fm:do>
  <fm:param-list>
    <fm:param name="MY_DATA" type="xs:string"/>
  </fm:param-list>
  <fm:return-list>
    <fm:return name="MY_OUTPUT" type="xs:string"/>
  </fm:return-list>
</fm:entry-theme>
```

In this examples entry theme, take no of:

- ***fm:-entry-theme*** type as ***service-rpc***; This tells the fox engine to build a WSDL document and listen to SOAP requests on this modules URL. To access the WSDL for a web service module the URL will follow the pattern http://.../fox/app_nmemonic/MODULE_NAME?WSDL

- MY_DATA as a parameter targetable as ***:{params}/MY_DATA***
- MY_OUTPUT as a return element targetable as ***:{return}/MY_OUTPUT***
- The processing of the specified parameter to the return data in ***fm:do***

As you can see, this looks to echo the parameter in ***:{params}*** to a return element in ***:{return}***. The difference between the param and return is that param will validate the input and return a ***SOAP:fault*** if the input data does not match up. ***fm:return*** is informational only, the fox module can return more than what is specified in ***fm:return-list*** but the WDSL will not show markup for anything not defined.

Below is a sample SOAP request to http://.../fox/app_nmemonic/MODULE_NAME

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:lc="http://www.myurl.co.uk/fox/webservices/devl/LC_WEBSRV">
   <soapenv:Header/>
   <soapenv:Body>
      <lc:GetMyTest>
         <MY_DATA>Echo what I say!</MY_DATA>
      </lc:GetMyTest>
   </soapenv:Body>
</soapenv:Envelope>
```

Below is a sample SOAP response for the above request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://www.myurl.co.uk/fox/webservices/devl/LC_WEBSRV">
   <soap:Header/>
   <soap:Body>
      <ns:GetMyTestResponse>
         <MY_OUTPUT>Echo what I say!</MY_OUTPUT>
      </ns:GetMyTestResponse>
   </soap:Body>
</soap:Envelope>
```

As you can now see, ***:{return}/MY_OUTPUT*** is now contained in the SOAP response.

Please note use use of:

```
<fm:entry-theme name="GetMyTest" type="service-rpc">
```

and the SOAP request markup

```
<lc:GetMyTest/>
```

When using ***fm:entry-theme*** type of service-rpc, this defines the SOAP document needed to access the correct webservice to be tightly specified. As you can see the entry theme and soap element share the same name (element to attribute). Meaning that when FOX is listening on the modules URL, it knows what SOAP data belongs to which service. There is another 'loose' method of doing this; by using type of ***service-document*** in entry theme instead as shown below.

```
<fm:entry-theme name="GetMyTest" type="service-document">
```

What this means is that FOX no longer needs to know which webservice this belongs too, which is fine in a single web service situation (single 'service-document scenario'). In the case of multiple service-document entry-themes' FOX will not know which to feed the SOAP data too. In order to circumvent this issue, Uniqie ***fm:param*** values for the root data element should be used. In most cases this is bad practice to define more than one ***service-document*** per module, ***service-rpc*** should be used instead.

## Notes

More information about SOAP and it's uses can be found here http://www.w3.org/TR/soap here http://www.w3schools.com/soap/soap_httpbinding.asp

When creating and testing SOAP request/response use of SoapUI is advised, this allows for SOAP document generation based off any WDSL and can be found here http://www.soapui.org/

## Related

- fm:entry-theme
- fm:param-list
- fm:return-list
- fm:return

# FOX:Module:param-list

## Description

***fm:param-list*** is part of FOX's web service integration, parameters described here are used for validation of a SOAP request. If the SOAP request is valid, FOX will process and return the result depending on the setup of ***fm:result-list***.

## Syntax

```
<fm:param-list>
  <fm:param name="Element Name" type="Element Type" xpath="XPath text" mand="Mand XPath" description="Description text"/>
  ...
</fm:param-list>
```

## Related

- fm:entry-theme
- fm:param
- fm:return-list

# FOX:Module:parameter

## Description

***fm:parameter*** is deprecated, use ***fm:param-list*** instead.

## Syntax

```
<fm:parameter>
  <fm:name/>
  <fm:description/>
  <fm:data-type/>
  <fm:data-mapping/>
</fm:parameter>
```

## Related

- fm:parameter-list
- fm:name
- fm:description
- fm:data-type
- fm:data-mapping

# FOX:Module:parameter-list

## Description

***fm:parameter-list*** is a deprecated feature of ***fm:documentation*** when a decendant of ***fm:entry-theme*** use ***fm:param-list*** instead.

## Syntax

```
<fm:parameter-list>
  <fm:parameter/>
</fm:parameter-list>
```

## Related

- fm:documentation
- fm:parameter

# FOX:Module:post-page

## Description

***fm:post-page*** is a child element of ***fm:pagination-definition***. This contain FOX Commands which will run ***AFTER*** a new page has been selected via ***fm:pagination-menu-out*** or ***fm:go-to-page*** and loaded into the DOM.

## Syntax

```
<fm:post-page>
  <fm:do>...</fm:do>
</fm:post-page>
```

## Examples

```
<fm:post-page>
  <fm:do>
    <fm:alert message="Showing string(:{sys}/sqlquery/paging/pagecount/text()) rows."/>
  </fm:do>
</fm:post-page>
```

The above example will return the count of rows returned into the DOM for the page.

## Related

- fm:pagination-definition

# FOX:Module:pre-condition

## Description

***fm:pre-condition*** is a child element of ***fm:documentation*** and is used storing notes about any parameters, contexts, or data required before use.

## Syntax

```
<fm:pre-condition>[ Pre-condition ]</fm:pre-condition>
```

## Examples

```
<fm:pre-condition>Parameter MY_ID must be set</fm:pre-condition>
```

## Related

- fm:documentaion

# FOX:Module:pre-page

## Description

***fm:pre-page*** is a child element of ***fm:pagination-definition***. This contain FOX Commands which will run after a new page has been selected via ***fm:pagination-menu-out*** or ***fm:go-to-page*** but ***BEFORE*** the results have been loaded into the DOM. Defining pre-page will alter how the pagination query returns results into the DOM, by default the query will ***PURGE*** the previous results, but with pre-page defined the query will now ***AUGMENT***. This allows for user control over which records should remain in the DOM, useful for holding user selected results from a previous query.

## Syntax

```
<fm:pre-page>
  <fm:do>...</fm:do>
</fm:pre-page>
```

## Examples

```
<fm:pre-page>
  <fm:do>
    <fm:remove match=":{root}/SEARCH_RESULTS/PRODUCTS[SELECT/SELECTED != 'true']"/>
  </fm:do>
</fm:pre-page>
```

The above example will remove any records from SEARCH_RESULTS where by they have not been selected, this behavior allows a user to select results to carry over from page to page (and query to query).

## Related

- fm:pagination-definition

# FOX:Module:presentation

## Description

The <fm:presentation> element is a container for <fm:set-page> and <fm:set-buffer> commands, which controls how the page is presented through HTML generation commands and standard HTML markup.

## Attribute Summary

*None*

## Examples

```
<fm:presentation>
  <fm:set-page>
    <html>
      <head>
        <title>FOX Basics</title>
      </head>
      <body>
        <fm:include name="buffer-content"/>
      </body>
    </html>
  </fm:set-page>
  <fm:set-buffer name="buffer-content">
    <h2>Hello World!</h2>
  </fm:set-buffer>
</fm:presentation>
```

## Related

- <fm:set-buffer>
- <fm:set-page>
- <fm:include>
- HTML Generation Command Reference

# FOX:Module:primary

## Description

***fm:primary*** is the container element for primary and foreign key references in table specifier in ***fm:table*** of which ***fm:primary*** is a child of.

## Syntax

```
<fm:primary>
  <fm:key>PRIMARY_KEY</fm:key>
  <fm:key>FOREIGN_KEY</fm:key>
  <fm:key>FOREIGN_KEY</fm:key>
  ...
</fm:primary>
```

## Examples

```
<fm:primary>
  <fm:key>PRODUCT_ID</fm:key>
  <fm:key>PRODUCT_TYPE</fm:key>
</fm:primary>
```

## Related

- fm:table
- fm:key

# FOX:Module:query

## fm:query

### Description

```
<fm:query name="String">
  [<fm:target-path match="'''Simple XPath'''"/>]
  [
    <fm:primary>
      [<fm:key>String</fm:key>...]
    </fm:primary>
  ]
  <fm:select>
    SQL Select Statement
  </fm:select>
  [<fm:using>...]
  [<fm:into>...]
</fm:query>
```

Defines a SQL select statement to be run with the run-query command. Bind variables can be defined via fm:using. If the query returns multiple rows they are returned into seperate XML complex types as defined by *target-path*.

## fm:target-path

The `match` attribute of this element provides a Simple XPath defining an element which will contain the column set for each row returned by this query. It is evaluated relative to the node which was targeted in the `match` attribute of the associated `fm:run-query` command.

Typically a target-path is defined on a query which will return multiple rows. When a target-path is evaluated, any elements which do not exist are initialised. The rightmost element in a target-path is initialised for every row. Because it is a Simple XPath, you cannot use Contexts within a target-path.

E.g. the target-path `./RESULT_LIST/RESULT` will initialise one element called RESULT_LIST which will contain as many RESULT elements as there are rows in the result set of the query. Each RESULT will contain the expected XML structure of a single row.

## fm:select

The SQL select statement is defined in the fm:select clause The select statement can optionally have Bind Variable placeholders Bind Variables can be bound by :name (preferred method), and by position number with :1 or ?

e.g. if using names, it does not matter which order you put them in when listing each <fm:using name="name">Xpath</fm:using>, however, if you use :3 for example in the query as a bind variable, you will have to make sure the third <fm:using>Xpath</fm:using> you make actually points to the element you require for that particular bind variable.

## fm:using

The fm:using clause is required for each bind variable Attribute name is only required when binding by name e.g. name=":data"

NB: Do not use any of the SQL reserved words as your bind variable names. I.E. :data

The value to bind is derived using XPath, either Attribute: datadom-location="/*/DATA" Text Content: <fm:using>:{params}/P_ACT_ID</fm:using> The XPath is evaluated relative to EACH Subject Element (in the run-query match clause) Optional attribute sql-type tells Oracle the SQL datatype provided. Default: VARCHAR2 Optional attribute datadom-type tells FOX the XMLSchema data type The default datadom-type is either obtained from the xs:element type or defaults to xs:string

## fm:storage-location Interaction

In addition to it's use in ***fm:run-query***, fm:query is used as part of ***fm:storage-location*** and behaves in a slightly different manner. ***fm:using*** binds can only be target by number not name (***:1, :2***) and the select DML fits inside ***fm:sql*** instead. See fm:storage-location for more advice.

## Related

- fm:database
- fm:storage-location
- fm:run-query
- fm:run-query2
- fm:using
- fm:target-path
- fm:select
- fm:into
- fm:row-lock
- fm:for-each-fetch

# FOX:Module:refresh-in-background

## Description

<fm:refresh-in-background> is a child element of <fm:map-set> and is used to control if the parent map-set should be refreshed in the background.

If the element contains 'false', then the map set will only be refreshed the next time the map-set is accessed and the timeout duration has expired. If 'true' then when the refresh timeout has expired, the map-set will be automatically refreshed.

Note that for background refreshing to be enabled, the refresh timeout must be at least 5 minutes.

## Attribute Summary

*None*

## Examples

```
<fm:map-set-list>
  <fm:map-set>
    <fm:storage-location/>
    <fm:do/>
    <fm:refresh-timeout-mins>10<fm:refresh-timeout-mins/>
    <fm:refresh-in-background>false</fm:refresh-in-background>
  </fm:map-set>
</fm:map-set-list>
```

## Related

- <fm:map-set>
- <fm:storage-location>

# FOX:Module:refresh-timeout-mins

## Description

<fm:refresh-timeout-mins> is a child element of <fm:map-set> and is used to control how often the map-set should be refreshed.

If the refresh timeout has expired, the next time the map set is accessed the contents will be reloaded by FOX.

The element contains an integer, which represents the refresh timeout in number of minutes. There is a special value 999999999, which instructs FOX to never refresh the map set - not even the first time.

## Attribute Summary

*None*

## Examples

```
<fm:map-set-list>
  <fm:map-set>
    <fm:storage-location/>
    <fm:do/>
    <fm:refresh-timeout-mins>10<fm:refresh-timeout-mins/>
    <fm:refresh-in-background>false</fm:refresh-in-background>
  </fm:map-set>
</fm:map-set-list>
```

## Related

- <fm:map-set>
- <fm:refresh-in-background>
- <fm:storage-location>

# FOX:Module:return

## Description

***fm:return*** is child element of ***fm:return-list*** and part of the web service processing in the FOX Engine using SOAP.

## Syntax

```
<fm:return name="Element Name" type="Element Content Type" xpath="XPath" mand="XPath" description="Text Description"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | XML Element Name | Name of XML element without namespace that is to be expected as input. | Yes |
| xpath | Xpath location | Location of xml in DOM to validate input against | No |
| type | xs:<data-type> or Element name | Datatype to validate against, element name can be used as data-type | Yes |
| mand | Xpath boolean | Describes if input XML element is mandatory | No |
| description | xs:string | Text description of parameter | No |

## Notes

This is a brief summary, most/all the inforamation regarding SOAP and ***fm:result***'s use in web services can be found at fm:param

## Related

- fm:entry-theme
- fm:param-list
- fm:return-list
- fm:param

# FOX:Module:return-list

## Description

***fm:return-list*** is a container for ***fm:return***, this is used as part of the FOX web services.

## Syntax

```
<fm:return-list>

  <fm:return name="Element Name" type="Element Type" xpath="XPath text" mand="Mand XPath" description="Description text"/>

  ...

</fm:return-list>
```

## Related

- fm:entry-theme
- fm:return
- fm:param-list

# FOX:Module:root-element

## Description

***fm:root-element*** contains the element name for ***:{root}*** to initialise to.

## Syntax

```
<fm:root-element>ROOT_ELEMENT_NAME</fm:root-element>
```

## Examples

```
<fm:root-element>FRUIT</fm:root-element>
```

Will initialise :{root} too

```
<FRUIT/>
```

## Notes

This will only initalise the ***:{root}*** DOM if ***fm:database***'s ***fm:select*** returns 0 rows, or is undefined.

## Related

- fm:new-document

# FOX:Module:row-lock

## Description

***fm:row-lock*** is a child element of ***fm:query***, this feature is now deprecated and ignored by the FOX engine.

## Related

- fm:query
- fm:select
- fm:current-of-item

# FOX:Module:schema

## Description

***fm:schema*** is a child element of ***fm:library-list***, the feature is deprecated and ignored. Use schemas defined inside ***fm:library*** instead.

## Related

- fm:library-list
- fm:library

# FOX:Module:security-list

## Description

<fm:security-list> is a container for <fm:mode-rule>, <fm:view-rule> and <fm:security-rule> commands and is used to implement security tables, a set of rules for overriding active namespaces for set-out/menu-out.
It is a further final way of filtering the content of the screen under different conditions.

Note that it is generally preferable to use a security table rather than rely on XPaths on individual set-out commands, as this greatly improves readability and maintainability

**Security rules work according to the following principles:**

1. Every namespace is turned on until mentioned in a security rule.
2. Once a namespace is mentioned in a rule it must be explicitly ENABLE-d (in that rule or another) if required.
3. A rule with operation ENABLE will whitelist the namespace under the specified conditions. Different ENABLE-ing rules are OR-ed together.
4. A mode-rule with operation DISABLE will blacklist the namespace under specified conditions. DISABLE-ing rules overwrite ENABLE-ing rules.
5. Attribute conditions are AND-ed together. e.g. theme="view" privilege="VIEW_OBJECT" means 'on entry theme "view" AND with privilege "VIEW_OBJECT")'
6. Attribute comma-separated-values are OR-ed together. e.g. theme="view, edit" privilege="VIEW_OBJECT" means '(on entry theme "view" OR "edit") AND with privilege "VIEW_OBJECT"'

This rule processing approach is similar to Fire Wall Rule Table processing.

## Attribute Summary

*None*

## Examples

*You will want to understand the <fm:mode-rule> and <fm:view-rule> commands before reading these examples.*

### Example 1

Consider that we have a *department-edit* namespace that we wish to disable in the state "state-view" we may try the security list:

```
<fm:security-list>
 <fm:mode-rule namespace="department-edit" operation="DISABLE" state="state-view"/>
</fm:security-list>
```

This will have the effect of turning off *department-edit* for state "state-view", but as *department-edit* has now been mentioned in a rule, it will be DISABLEd for all conditions (principle 2 above).

Thus we have to enable the namespace for other conditions:

```
<fm:security-list>
 <fm:mode-rule namespace="department-edit" operation="DISABLE" state="state-view"/>
 <fm:mode-rule namespace="department-edit" operation="ENABLE"/>
</fm:security-list>
```

Now consider that we want to only allow *department-edit* where the user has the privilege "EDIT_DEPARTMENT"; we can do this:

```
<fm:security-list>
 <fm:mode-rule namespace="department-edit" operation="DISABLE" state="state-view"/>
 <fm:mode-rule namespace="department-edit" operation="ENABLE" privilege="EDIT_DEPARTMENT"/>
</fm:security-list>
```

**N.B.** If a user enters the state "state-view" with privilege "EDIT_DEPARTMENT" then *department-edit* will be DISABLEd. Due to principle 4 processing for the namespace stops after a successful DISABLE.

### Example 2

Now consider we are scrapping the namespace *department-edit* and wish to use a namespace *department* instead. We want to control whether fields are read-only or editable via a security table and try:

```
<fm:security-list>
 <fm:mode-rule namespace="department" operation="ENABLE"/>
 <fm:view-rule namespace="department" operation="ENABLE" theme="view"/>
</fm:security-list>
```

The mode-rule ENABLE-s *department* as editable for all conditions initially.
The developer has then attempted to force *department* to be read-only under the entry-theme "view" using a view-rule.
Due to principal 3 above, however, the ENABLE-ing rules are OR-ed together and the mode-rule makes the view-rule redundant.
The developer may solve this by doing the following:

```
<fm:security-list>
 <fm:mode-rule namespace="department" operation="ENABLE" theme="edit, create"/>
 <fm:view-rule namespace="department" operation="ENABLE" theme="view"/>
</fm:security-list>
```

Thus allowing *department* fields to be editable on only the "edit" and "create" entry themes, and read-only on "view".

We next want to add a condition so that nobody can view/edit data without the "DEPARTMENT" privilege:

```
<fm:security-list>
 <fm:mode-rule namespace="department" operation="ENABLE" theme="edit, create" privilege="DEPARTMENT"/>
 <fm:view-rule namespace="department" operation="ENABLE" theme="view" privilege="DEPARTMENT"/>
</fm:security-list>
```

The privilege "DEPARTMENT" condition must be added to both rules. Both rules will be evaluated and can ENABLE *department* even if the first fails.

## Related

- <fm:mode-rule>
- <fm:view-rule>
- <fm:security-rule>

# FOX:Module:security-rule

## Description

A set of conditions (privilege, state, theme and XPath) may be defined using an **<fm:security-rule>**.
These can be referenced using the *rule-ref* attribute on another mode or view-rule.
This allows complicated mode attributes to be defined once, and then referenced in multiple rules.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | Literal String | The name of the security-rule. This name can then referenced in the 'rule-ref' attribute of a mode/view-rule. | Yes |
| privilege | Literal String | The privilege(s) the user must have to activate the mode/view-rule that references this security-rule. Can be a comma separated list. | No |
| state | Literal String | The sate(s) the user must be in to activate the mode/view-rule that references this security-rule. Can be a comma separated list. | No |
| theme | Literal String | The entry theme(s) the user must be in to activate the mode/view-rule that references this security-rule. Can be a comma separated list. | No |
| xpath | XPath String | The XPath test run relative to the root element. If it evaluates to true, the mode/view-rule that references this security-rule is activated. | No |

### Examples

The following will apply the privilege="DEPARTMENT" state="licence-review" attributes to both the mode-rule and view-rule:

```
<fm:security-list>
 <fm:security-rule name="department-privs" privilege="DEPARTMENT" state="licence-review"/>
 <fm:mode-rule namespace="department" operation="ENABLE" rule-ref="department-privs"/>
 <fm:view-rule namespace="department" operation="ENABLE" theme="view" rule-ref="department-privs"/>
</fm:security-list>
```

### Related

- <fm:mode-rule>
- <fm:view-rule>
- <fm:security-list>

# FOX:Module:select

### Description

***fm:select*** is a child element of ***fm:query*** and contains an single SQL statement which will run upon ***fm:run-query*** is called. ***fm:using*** allows for bind variables, targeting data in a DOM, inside the select statement

### Syntax

```
<fm:select>
[ SQL SELECT STATMENT ]
</fm:select>
```

### Examples

```
<fm:select>
SELECT
    product_id
  , product_name
  , prouct_price
FROM prodmgr.products
WHERE product_id = :product_id
</fm:select>
<fm:using name=":product_id">:{params}/PRODUCT_ID</fm:using>
```

## Notes

It's considered good practice to align SQL statement to the left of the module XML, PrettyPrint functions can often be configured to ignore formatting on the contents of specific tags. This allows other developers to copy and paste the SQL into Toad without modification or hassle and is generally easier to read.

## Related

- fm:query
- fm:using

# FOX:Module:set-buffer

## Description

The <fm:set-buffer> element is a container for all the HTML generation commands you wish FOX to run, plus any additional HTML which may be needed for boilerplate text, page structure, etc. IN other words, it is the definition of the presentation markup for a logical area of the screen. FOX allows most standard HTML markup within buffers (and <fm:set-page>), with a few exceptions such as <script> tags. Using buffers allows presentation logic to be reused between modules and states, and also helps to keep the <fm:presentation> section more logically structured. As a result, buffers are one of the most common libraried in elements.

The convention is to wrap all buffers in a top-level buffer with a name attribute of 'buffer-content'. This makes librarying in <fm:set-page> and layout templates possible, as they respect this naming convention and will always include 'buffer-content'.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | Literal String | The name of the buffer. Used when referencing the buffer from an <fm:include> command. | Yes |
| stub-overload | XPath | If the Xpath evaluates to true, any libraried in <fm:set-buffers> with the same name will replace this one. See <fm:library>. | No |

## Examples

```
<fm:presentation>
<fm:set-page>
  <html>
    <head>
      <title>FOX Basics</title>
    </head>
    <body>
      <fm:include name="buffer-content"/>
    </body>
  </html>
</fm:set-page>
```

```
  <fm:set-buffer name="buffer-content">
    <h2>Hello World!</h2>
  </fm:set-buffer>
</fm:presentation>
```

### Related

- <fm:presentation>
- <fm:set-page>
- <fm:include>
- HTML Generation Command Reference

# FOX:Module:set-page

## Description

<fm:set-page> is the top-level command which FOX looks for to decide what to display to the user. It will typically contain the basic skeleton of an HTML page such as <html>, <head> and <body> tags. Although not mandatory, the set-page element should always include 'buffer-content' as this is the name used for the top-level buffer which contains the page layout commands. Often the set-page element is imported, typically from the LAYOUT1 module.

FOX allows most standard HTML markup within buffers (and <fm:set-page>), with a few exceptions such as <script> tags.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| stub-overload | XPath | If the Xpath evaluates to true, any libraried in <fm:set-buffers> with the same name will replace this one. See <fm:library>. | No |

## Examples

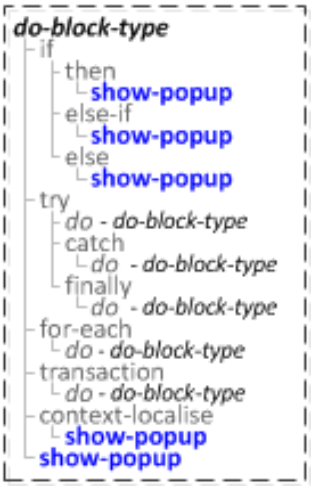
```
  <fm:presentation>
  <fm:set-page>
    <html>
      <head>
        <title>FOX Basics</title>
      </head>
      <body>
        <fm:include name="buffer-content"/>
      </body>
    </html>
  </fm:set-page>
  <fm:set-buffer name="buffer-content">
    <h2>Hello World!</h2>
  </fm:set-buffer>
</fm:presentation>
```

## Related

- <fm:presentation>
- <fm:set-buffer>
- <fm:include>
- HTML Generation Command Reference

# FOX:Module:show-popup

## Schema Location


## Description

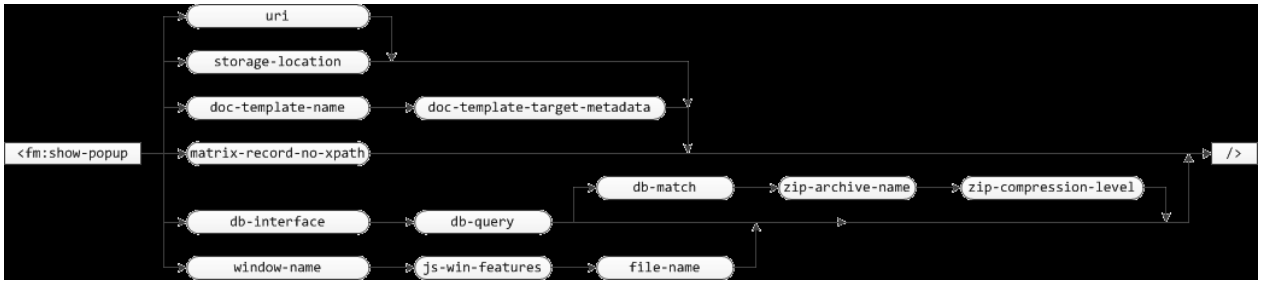
Multifunctional command which uses a browser popup window to display either:

- An arbitrary URI
- The contents of a storage location
- A FOX-generated PDF document
- A set of Matrix records
- A streamed zip file containing multiple files

This will often result in a file download being initiated by the browser.

NOTE: due to a bug in IE6/7/8 and systems with restricted settings, show-popup commands which initiate a file download require low system privilege users to depress the CTRL key from initiating the action to seeing the save file dialog.

## Syntax



## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| uri | XPath String | Pops up a window based on the resolved uri. | No |
| storage-location | String Literal | Pops up a window based on the given storage location and file name. | No |
| doc-template-name | XPath String | The name of the document template to use in generating the pdf. | No |
| doc-template-target-metadata | Single Node Complex XPath | The node to put the processed template metadata into. | Only when `doc-template-name` is used. |
| matrix-record-no-xpath | Single Node Complex XPath | A list of nodes containing matrix record numbers. | No |
| db-interface | String Literal | The db-interface to use for multiple file query to zip. | No |
| db-query | String Literal | The db-query to use for multiple file query to zip. **Please Note:**<br>• The query specified *must* use the datadom-location attribute for XPath binds. The syntax `<fm:using>...</fm:using>` is currently not supported.<br>• Contexts (i.e. " :{theme} [1]") are supported from version 4.04.15 [2] onwards, and are executed relative to the `db-match` location.<br>• The query MUST yield, at least, a column named "BLOB" **or** "CLOB" and the column "FILENAME". Optional columns are "MIMETYPE" and "PATH" (relative path within ZIP file). | Only when `db-interface` is used. |
| db-match | String Literal | The match path to use for multiple file query to zip. **Default:** "." | No |
| zip-archive-name | XPath String | The name to use for the zip archive that will be offered to the user to download. **Default:** "Download.zip" | Only when `db-match` is used. |
| zip-compression-level | XPath Integer | The compression level to use, must be between 0-9 or -1. **Default:** -1 | Only when `db-match` is used. |
| window-name | 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 |
| js-win-features | String Literal | A comma-separated list of properties for the child window that will display the popup. 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"` | Only when `window-name` is used. |
| file-name | String Literal | Pops up a window based on the given storage location and filename. | Only when `window-name` is used. |

## Examples

```
<fm:show-popup
    [uri="XPath"]
  | [storage-location="string" ]
  | [doc-template-name="XPath" doc-template-target-metadata="XPath"]
  | [matrix-record-no-xpath="XPath"]
```

```
    | [db-interface="string" db-query="string" [db-match="string"
zip-archive-name="XPath" zip-compression-level="XPath"]]
    [window-name="string" js-win-features="string" file-name="string" ]
  />
```

# FOX:Module:sql

## Description

***fm:sql*** contains SQL DML statements in conjunctions with storage locations, these statements do not use named binds and as such need numbered binds for each ***fm:using*** present instead (***:1, :2***)

## Syntax

```
<fm:sql>
[ SQL DML Statement ]
</fm:sql>
```

## Examples

***fm:query***, ***fm:update***, ***fm:insert*** and ***fm:delete*** all contain ***fm:sql*** as a sub element to hold SQL DML statements

```
<fm:sql>
SELECT xml_data FROM portal_folders WHERE id = :1
FOR UPDATE OF xml_data NOWAIT
</fm:sql>
```

## Note

Use of ***fm:insert*** is generally discouraged, the same affect can be accomplished through the use of ***fm:api*** in a more controlled manner.

## Related

- fm:query
- fm:update
- fm:insert
- fm:delete

# FOX:Module:state

## Description

***fm:state*** is a child of ***fm:state-list***, this provides callable presentation and action markup which are scoped to the state of which they belong. Which means the presentation markup and actions are only available to be called if the current state matches the state of which those elements are defined.

States can be pushed onto the state stack, replaced, and popped off the state stack. They separate distinguishable areas of your fox module, this usually consists of different screens that the user will see and interact with. One example states can be used for is to separate different sections of a large application form, allowing for markup and actions used in 'Section 1' to be separated with 'Section 2' and so forth. This is just one basic use, many FOX modules use a multitude of pushing and popping states to achieve their target functionality.

## Syntax

```
<fm:state name="[ State Name ]" title="[ State Title ]" description="[ State Description ]">
  <fm:documentation>...</fm:documentation>
  <fm:action-list>...</fm:action-list>
  <fm:presentation>...</fm:presentation>
</fm:state>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | xs:string | Name of state, prefixed 'state-' (i.e ***state-manage*** ) | Yes |
| title | xs:string | Human readable title of the state | No |
| description | xs:string | Short desscription of the state. | No |

## Examples

The following example shows some important properties of states.

```
<fm:state-list>
  <fm:state name="state-initial" title="Initail State" description="Initial state of the module">
    <fm:action-list>
      <fm:action name="action-state-push" fox:run=".">
        <fm:do>
          <fm:state-push name="state-demo"/>
        </fm:do>
      </fm:action>
      <fm:action name="action-state-replace" fox:run=".">
        <fm:do>
          <fm:state-replace name="state-demo"/>
        </fm:do>
      </fm:action>
    </fm:action-list>
    <fm:presentation>
```

```
      <fm:set-buffer name="buffer-content">
        <fm:action-out name="action-state-push" fox:mode="."/>
        <fm:action-out name="action-state-replace" fox:mode="."/>
      </fm:set-buffer>
    </fm:presentation>
  </fm:state>
  <fm:state name="state-demo" title="Demo States" description="Demonstates the use of states">
    <fm:action-list>
      <fm:action name="action-message-name" fox:run=".">
        <fm:do>
          <fm:alert message="Hi! you're in state: string(:{sys}/state/name/text())" />
        </fm:do>
      </fm:action>
      <fm:action name="action-message-title" fox:run=".">
        <fm:do>
          <fm:alert message="Hi! you're in state: string(:{sys}/state/title/text())" />
        </fm:do>
      </fm:action>
      <fm:action name="action-pop-state" fox:run=".">
        <fm:do>
          <fm:state-pop/>
        </fm:do>
      </fm:action>
    </fm:action-list>
    <fm:presentation>
      <fm:set-buffer name="buffer-content">
        <fm:action-out action-out="action-message-name" fox:mode="."/>
        <fm:action-out action-out="action-message-title" fox:mode="."/>
        <fm:action-out action-out="action-pop-state" fox:mode="."/>
      </fm:set-buffer>
    </fm:presentation>
  </fm:state>
</fm:state-list>
```

This is the entry-theme for the above example.

```
<fm:entry-theme name="new">
  <fm:storage-location>sl-main</fm:storage-location>
  <fm:state>state-inital</fm:state>
  <fm:attach>/*</fm:attach>
  <fm:do/>
</fm:entry-theme>
```

The example above shows two states, ***state-initial***, which is the first state that the module loads into as shown by the entry-theme example, and ***state-demo***. ***state-initial*** will show two links, one for each of the state stack controlling commands:

- ***action-state-push*** pushes ***state-demo*** onto the call-stack, the presentation and action elements will take highest precedence and replace other buffers. This effectively becomes the state the user will now see and interact with.

- ***action-state-replace*** will do the same as ***fm:state-push***, however will not add to the state stack but rather replace the current state with the selected state.

***state-demo*** shows how the title and name can be fetched from the ***:{sys}*** DOM and how state's can be popped the state-stack to return to the previous state.

- ***action-message-name*** shows how the states internal name can be targeted and set out into a JavaScript alert.
- ***action-message-title*** show the same functionality as ***action-message-name*** but the human readable form.
- ***action-pop-state*** will pop the current state of the state stack, this works if the previous state was loaded using ***fm:push-state***. However if the state was loaded using ***fm:replace***, unexpected results can occur. The state previous to the current state (which replaced the previous state) will become the active state, if no previous state exists (like in this example) then a FOX error will be raised.

## Related

- fm:state-list
- fm:state-push
- fm:state-pop
- fm:state-replace
- fm:presentation
- fm:action-list

# FOX:Module:state-list

## Description

***fm:state-list*** is a module level element which acts as a container for repeating ***fm:state*** elements.

## Syntax

```
<fm:state-list>
  <fm:state name="[ State Name ]">...</fm:state>
  ...
</fm:state-list>
```

## Related

- fm:module
- fm:state

# FOX:Module:statement

## Description

***fm:statement*** is a child element of ***fm:api*** and contains an anonymous block of PL/SQL which will get executed upon calling the command ***fm:run-api***.

## Syntax

```
<fm:statement>
[ Anonymous PL/SQL block ]
</fm:statement/>
```

## Examples

Below is an example of ***fm:statement*** containing an anonymous PL/SQL block being used in conjunction with ***fm:using*** to read out the sysdate to a temporary location

```
<fm:api name="api-sysdate">
  <fm:statement>
DECLARE
  l_sysdate VARCHAR2(4000);
BEGIN
  SELECT sysdate()
  INTO l_sysdate
  FROM dual;
  :sysdate := l_sysdate;
END;
  </fm:statement>
  <fm:using name=":sysdate" datadom-type="string" sql-type="varchar" direction="out">:{temp}/SYDATE</fm:using>
</fm:api>
```

## Notes

It's considered good practice to align PL/SQL statements to the left of the module XML, PrettyPrint functions can often be configured to ignore formatting on the contents of specific tags. This allows other developers to copy and paste the PL/SQL into Toad without modification or hassle and is generally easier to read.

## Related

- fm:api
- fm:run-api

# FOX:Module:storage-location

## Description

Storage Locations provide FOX with a standard mechanism for accessing/updating units of data. They are a definition of how and where data is stored and accessed. They differ from ***fm:query*** definitions as Fox decides where, when, and how to use the SQL/DML defined within the Storage Location.

Within the storage location element, you can add an ***fm:database*** node, to tell FOX that the data from the Data DOM has a direct relationship with a database table. The Data DOM would be stored as a Clob or XML column on this table.

The ***fm:database*** syntax has 3 child elements:

- ***fm:query*** in which you can write SELECT statements
- ***fm:update*** in which you can write UPDATE statements
- ***fm:insert*** in which you can write INSERT statements (This is rarely used, most modules will create new records separately)

## Syntax

```
<fm:storage-location name="name" xml-storage-type="clob">
  <fm:cache-key string="module name :bind">
    <fm:using>bind xpath</fm:using>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>root</fm:root-element>
  </fm:new-document>
  <fm:database>
    <fm:query>
      <fm:sql>
SQL statement with :bind
      </fm:sql>
      <fm:using>bind xpath</fm:using>
    </fm:query>
    <fm:insert>
      <fm:sql>
DML statement with :bind
      </fm:sql>
      <fm:using>bind xpath</fm:using>
    </fm:insert>
    <fm:update>
      <fm:sql>
DML statement with :bind
      </fm:sql>
      <fm:using>bind xpath</fm:using>
    </fm:update>
  </fm:database>
</fm:storage-location>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | String | Name of the storage location, this should have a prefix of 'sl' to denote storage location (i.e sl-search-results). | Yes |
| xml-storage-type | String | `clob` or `binary`. Set this attribute based on the underlying storage your XML is being written to. (E.g. based on the Oracle STORE AS clause). If you do not set this to `binary` and the XML is stored as binary, this will cause problems because the Oracle XML serialiser introduces extra whitespace which FOX is not expecting. By setting the attribute correctly, the whitespace issue is solved by setting the `xml:space` attribute on the document's root element to "preserve". | No |

## Concepts

A global caching mechanism is used across the FOX System. Data is loaded into the memory cache bucket, which is identified using a cache key (I.E. the name of the bucket). All operations are performed on the data held in the memory cache as this provides best performance. Because the caching mechanism is global across FOX, two different user sessions, running different modules can access a common unit of data - simply by addressing the same bucket (I.E. using the same cache key). This means both users can benefit from seeing the common data consistently and quickly, whilst system memory is optimised as only one version of the data is held.

You can have multiple storage location definitions within a module, but each one should have a unique cache key and name.

Across modules, the cache key must be unique for discrete data. Common data can however be reused by engineering a repeatable cache key.

- Storage Location is a definition of how and where something is stored
  - Defines all database SQL and DML to access a single row of information
  - Complex internal caching aids performance and data sharing
- When a module is loaded, a cache key is generated. This is used as a name of a memory location to store retrieved information for that module
- It is important to get this correct, as two modules with the same storage location cache key will use the same memory location (bucket). If different data structures are involved, this is very bad as the two data structures may be corrupted
- A cache key should include some relevant text as well as either a unique bind variable or a bind variable that has been passed into the module
- The cache key for most modules will be the module name, and some bind variable
- Storage locations are used for accessing:
  - Short term computed lists of values (map-sets, covered next)
  - Static lists of values (map-sets)
  - Module Data DOM – XML stored in database
  - Files stored on the database (e.g. Uploaded Word files)

## Examples

An example of a storage location for a module called MOD012X that had no variables passed into it and would use a unique bind variable is:

```
<fm:storage-location name="main">
  <fm:cache-key string="MOD012X :1"> '''(This ':1' is replaced by a unique value populated by FOX)'''
    <fm:using using-type="UNIQUE"/>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>ROOT</fm:root-element>
  </fm:new-document>
</fm:storage-location>
```

For the same module, but with a variable called ID passed into the module, you would use the following code to declare the storage location:

```
<fm:storage-location name="main">
  <fm:cache-key string="MOD012X :1">
    <fm:using>:{params}/ID</fm:using>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>ROOT</fm:root-element>
  </fm:new-document>
</fm:storage-location>
```

Here is an example of a storage location being used to keeping the ***:{root}*** DOM synchronized to the table ***portal_folders*** for a particular ***:{params}/P_PF_ID*** value.

```
<fm:storage-location name="main">
  <fm:cache-key string="MODULE_NAME:1">
    <fm:using>:{params}/P_PF_ID</fm:using>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>FOLDER</fm:root-element>
  </fm:new-document>
  <fm:database>
    <fm:query>
      <fm:sql>
SELECT xml_data FROM portal_folders WHERE id = :1
FOR UPDATE OF xml_data NOWAIT
      </fm:sql>
      <fm:using>:{params}/P_PF_ID</fm:using>
    </fm:query>
    <fm:update>
      <fm:sql>
UPDATE portal_folders SET id = id WHERE id = :1
      </fm:sql>
      <fm:using>:{params}/P_PF_ID</fm:using>
    </fm:update>
```

```
  </fm:database>
</fm:storage-location>
```

As you can see, ***fm:cache-key*** is using a value passed in from ***:{params}*** to creation a memory location unique to this module and parameter value used for fast access of ***:{root}*** which in turn allows for faster query/updates from the database.

***fm:new-document*** is only used/triggered when ***fm:database***'s select statement fails to return a row or ***fm:database*** is abscent all together. ***:{root}*** is then initialised as the as ***fm:root-element***'s value. In the above example this would be

```
<FOLDER/>
```

***fm:query*** is the select statement which will initialise ***:{root}*** with the returned xml data. So if the data returned was

```
<FOLDER>
  <DOCUMENT>Banana</DOCUMENT>
  <DOCUMENT>Orange</DOCUMENT>
</FOLDER>
```

***:{root}*** Would be bound to the root element of the return data, which is ***<FOLDER>*** and to return all the document elements in folder the xpath would be ***:{root}/DOCUMENT***

***fm:update*** is called whenever a change to the ***:{root}*** occurs, this is often used to fire update triggers which are missed due LOB's not firing on updates.

## Warnings and Caveats

### Storage Location / Entry Theme execution order

The storage location cache key is evaluated **before** the entry-theme's **<fm:do/>** block. This means that you should only base a storage location cache key on DOM values that are available prior to entry-theme processing. In practise, this limits you to using :{params}, :{env}, :{session} and :{user} - as no other DOMs have useful data for cache-keys at that particular point of code execution.

Sometimes people make the mistake of augmenting data in the entry-theme and using the resultant value in the cache key. Here is a simplified example.

```
<fm:storage-location name="main">
  <fm:cache-key string="MOD012X :1">
    <fm:using>:{theme}/DETAIL_ID</fm:using>
  </fm:cache-key>
  <fm:new-document>
    <fm:root-element>ROOT</fm:root-element>
  </fm:new-document>
</fm:storage-location>
```

1. When the module is called, the cache-key evaluates *immediately* and resolves to "MOD012X " (the XPath substituted as :1 has returned no data). Note that if multiple callers attempt to access the module at the same time, there will be locking contention, as they are all sharing the exact same cache key.
2. The entry-theme **<fm:do/>** block executes, and populates :{theme}/DETAIL_ID. This results in a different id for each entry.
3. On the next FOX action click, each user's cache key will evaluate to a unique string, for example "MOD012X 13" and "MOD012X 14", as the :{theme}/DETAIL_ID element is populated.

While this seems innocuous enough, the locking contention can cause problems, especially if the entry-theme processing takes a long time to complete. Please be aware of the general principle that cache keys should be based on values available at the point of entry to a module.

### Explicit DOM binding

Using the `<fm:using using-type="DATA-XMLTYPE"/>` bind will cause FOX to write the root DOM's XML to the storage location twice, as it is already streamed straight into the LOB locator by default. Therefore it is recommended that you **do not use the `DATA-XMLTYPE` bind**, and only use the update statement to set other columns, such as last update times, etc.

If your storage location XML has XViews on it, an update statement still needs to be executed in order to cause the XView triggers to fire. A self-update (i.e. *SET id = id*) will be sufficient for this.

The exception to the above rule is that for binary XML storage locations, the `DATA-XMLTYPE` bind **SHOULD** be used due to a bug with the current JDBC, whereby the DOM is NOT streamed back to the LOB locator. This should be fixed in a future FOX release. Errors such as this are symptomatic of this error:

```
Storage Location: Update/Query pair do not access the same row/column
(change number inconsistant)
```

Here is an example of how to fix the problem:

| Broken | Fixed |
|---|---|
| `<font size="8.35">`<br>`<fm:update>`<br>`  <fm:sql>`<br>`UPDATE trainingmgr.xx_plan_app_details`<br>`SET last_updated_by = :1`<br>`WHERE application_id = :2`<br>`AND status_control = 'C'`<br>`  </fm:sql>`<br>`  <fm:using>:{user}/WUA_ID</fm:using>`<br>`  <fm:using>:{params}/APPLICATION_ID</fm:using>`<br>`</fm:update>`<br>`</font>` | `<font size="8.35">`<br>`<fm:update>`<br>`  <fm:sql>`<br>`UPDATE trainingmgr.xx_plan_app_details`<br>`SET xml_data = :1`<br>`, last_updated_by = :2`<br>`WHERE application_id = :3`<br>`AND status_control = 'C'`<br>`  </fm:sql>`<br>`  <fm:using using-type="DATA-XMLTYPE"/>`<br>`  <fm:using>:{user}/WUA_ID</fm:using>`<br>`  <fm:using>:{params}/APPLICATION_ID</fm:using>`<br>`</fm:update>`<br>`</font>` |

## Related

- fm:storage-location-list
- fm:cache-key
- fm:new-document
- fm:database

# FOX:Module:storage-location-list

## Description

***fm:storage-location-list*** is a container for ***fm:storage-location*** and ***fm:file-storage-location*** child elements.

## Syntax

```
<fm:module>
  <fm:storage-location-list>
    <fm:storage-location>...</fm:storage-location>
    <fm:file-storage-location/>...<fm:file-storage-location>
  </fm:storage-location-list>
</fm:module>
```

## Notes

If both ***fm:storage-location*** and ***fm:file-storage-location*** are contained in ***fm:storage-location-list*** then all ***fm:storage-location*** child elements must appear before the ***fm:file-storage-location*** elements.

## Related

- fm:storage-location
- fm:file-storage-location

# FOX:Module:table

## Description

***fm:table*** defines rules for insert update and deletes based of an XML tree structure that represents a row in the table. Basic table information is described as attributes on the root tag, child elements contain information on specific columns and their relationship identifiers (PRIMARY KEY etc). Based on the root element that ***fm:run-dml*** matches too and the ***rec*** element specified within, this statement will either update the table with the new records, remove the records or insert new records. This is considered legacy code and should not be used in the future, ***fm:api*** is a much better approach.

## Examples

```
<fm:table name="TABLE NAME" cols="No. of columns" [ins="Insert flag" upd="Update flag" del="Delete flag" lock-when="Lock statment" del-when="Delete statement" namespace="Namespace"]>

  <fm:map-path match="XPath String"/>

  <fm:primary>

    <fm:key>Table Column Key</fm:key>

  </fm:primary>

  <fm:using>XPath Bind</fm:using>#

  <fm:for-each/>

</fm:table>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | fm:entered-string | Indentifies the database table to lock and/or maintain | Yes |
| cols | xs:positiveInteger | a check sum attribute which must match the number of columns included in the DML interface. It is hoped the module specifier computes cols from requirements, rather than counting up elements with {interface}=. attributes in the module specification | Yes |
| ins | fm:yn-flag | Insert priviledge allowed | No |
| ins | fm:yn-flag | Update priviledge allowed | No |
| del | fm:yn-flag | Delete priviledge allowed | No |
| lock-when | xs:string | In addition to the DML match tag and the rec-path match tag this tag specifies extra XML search conditions to further restrict records for processing | No |
| del-when | xs:string | In additions to the DML match tag and the rec-path match tag this tag specifies extra XML search conditions to further restrict records for processing | No |
| namespace | fm:entered-string | Corresponds to ns in the ns:cols attribute of a fox element definition (to map dom columns to database columns) | No |

## Examples

```
<fm:db-interface name="dbint-products">
  <fm:table name="PRODUCT_DETAILS" cols="4" ins="N" upd="Y" del="N" >
    <fm:map-path/>
    <fm:primary>
      <fm:key>PRODUCT_ID</fm:key>
    </fm:primary>
  </fm:table>
</fm:db-interface>
```

Note that the use of ***fm:table*** is unique to this interface, only one ***fm:table*** can be contained inside, as demonstated by the syntax of ***fm:run-dml***

```
<fm:run-dml interface="dbint-products" upd="Y" match=":{theme}PRODUCT_INFO/PRODUCT"/>
```

## Related

- fm:db-interface
- fm:primary
- fm:map-path
- fm:key
- fm:run-dml

# FOX:Module:target-path

## Description

***fm:target-path*** defines the DOM target for all rows returns by ***fm:query***. Mandatory for all queries returning more then one row.

## Syntax

```
<fm:target-path match="XPath to DOM target"/>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| Match | XPath location of DOM | Location for output of each record returned by the query, location is relative to run-query. More often than not, ***fm:run-query*** will match on a list node (:{theme}/PRODUCT_LIST) ***fm:target-path*** is then matched to individual elements of said query (PRODUCT). | Yes |

## Examples

Given this ***fm:run-query*** command.

```
<fm:run-query interface="dbint-products" query="qry-get-products" match=":{temp}/PRODUCT_LIST"/>
```

and this ***fm:target-path*** of the query.

```
<fm:taget-path match="PRODUCT"/>
```

The resulting DOM woud look as follows.

```
<PRODUCT_LIST>
  <PRODUCT>Child elements with Data...</PRODUCT>
  <PRODUCT>Child elements with Data...</PRODUCT>
  <PRODUCT>Child elements with Data...</PRODUCT>
</PRODUCT_LIST>
```

## Related

- fm:query
- fm:run-query

# FOX:Module:template

## Description

<fm:template> is used to create a general XML structure which can then be referenced and initialized anywhere in the module. It is commonly used to generate <fm:map-set> key-data XML.

A template can also be initialised using <fm:init>

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| name | Literal String | The name of the Template, used as a reference. | Yes |

## Examples

```
<fm:template-list>
  <fm:template name="tmpl-ms-gender">
    <map-set>
      <rec>
        <key>Male</key>
        <data>m</data>
      </rec>
      <rec>
        <key>Female</key>
        <data>f</data>
      </rec>
      </rec>
    </map-set>
  </fm:template>
</fm:template-list>
```

## Related

- <fm:map-set>

# FOX:Module:template-list

## Description

<fm:template-list> is a container for <fm:template> elements, which are used to create XML templates.

## Attribute Summary

*None*

## Examples

```
<fm:template-list>
  <fm:template name="tmpl-ms-gender">
    <map-set>
      <rec>
        <key>Male</key>
        <data>m</data>
      </rec>
      <rec>
        <key>Female</key>
        <data>f</data>
      </rec>
      </rec>
    </map-set>
  </fm:template>
</fm:template-list>
```

## Related

- <fm:template>
- <fm:map-set>

# FOX:Module:title

## Description

***fm:title*** is part of ***fm:header*** and allows for setting out text which describes your specific fox module. The text entered is populated as part of the ***:{sys}*** DOM under the heading ***title***. This in turn is extracted by the ***LAYOUT1LIB*** library module, added to a buffer and included as part of your module's html markup.

## Syntax

```
<fm:header>
  <fm:title>(Module title text)</fm:title>
</fm:header>
```

## Examples

Given this as the state of the module.

```
  <fm:header>
    <fm:title>My Module Title</fm:title>
    ...
  </fm:header>
```

This would be the result in ***:{sys}*** when accessing that module.

```
<module>
  <title>My Module Title</title>
  ...
</module>
...
```

This in turn would show on screen as a header with the text contents of ***fm:title*** through ***LAYOUT1LIB***'s formatting

Which is effectively doing this:

```
<fm:set-buffer name="buffer-title2">
  <fm:expr-out match=":{sys}/module/title"/>
</fm:set-buffer>
```

This buffer is then included in an appropriate part of the page.

## Notes

***LAYOUT1LIB*** Is a formatting library, therefore may be subject to change (or not used at all in your module).

## Related

- fm:header

# FOX:Module:update

## Description

***fm:update*** describes a DML update statement which is triggered upon a change to ***:{root}***. This is typically used to kick off database level triggers that updates to LOB data miss.

## Syntax

```
<fm:update>
  <fm:sql>
[ SQL Update Statement ]
  </fm:sql>
  <fm:using>:bind xpath</fm:using>
</fm:update>
```

## Examples

This is a typical update statement used to kick off triggers not targeted by LOB updates. Setting a column to the value of itself for the locked row (as targeted by the bind).

```
<fm:update>
  <fm:sql>
UPDATE portal_folders SET id = id WHERE id = :1
  </fm:sql>
  <fm:using>:{params}/P_PF_ID</fm:using>
</fm:update>
```

There is no need to add in a using-type="DATA-XMLTYPE" bind for the record to be updated with the contents of the DOM, FOX knows which column to target through the ***fm:select*** statement and automatically posts changes from the DOM to the record. Use of using-type="DATA-XMLTYPE" will duplicate ***:{root}*** in storage.

```
<fm:update>
  <fm:sql>
UPDATE portal_folders SET id = id WHERE id = :1
  </fm:sql>
  <fm:using>:{params}/P_PF_ID</fm:using>
  <fm:using using-type="DATA-XMLTYPE"/>
</fm:update>
```

## Related

- fm:database
- fm:using
- fm:sql

# FOX:Module:using

## Description

***fm:using*** appears across a range of FOX ***FM:*** module schema markup. It's typical use is to bring in data from a DOM via the use of bind variables into a PL/SQL or SQL statement. It's exact use is specific to the context in which it is used.

## Syntax

When ancestor is ***fm:storage-location-list***

```
<fm:using using-type="Using Type">bind XPath location</fm:using>
```

When ancestor is ***fm:db-interface-list***

```
<fm:using name="Bind name" datadom-location="Bind XPath location" sql-type="SQL Type" datadom-type="Datadom Type">bind XPath location</fm:using>
```

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| using-type | • XPATH<br>• UNIQUE<br>• STATIC<br>• DATA-XMLTYPE<br>• DATA-CLOB<br>• DATA-BLOB<br>• FILE-METADATA-XMLTYPE | Type used for context of data passed through ***fm:using*** | No |
| name | xs:string | Name of bind (i.e :document_id) | Yes |
| datadom-location | XPath String | XPath location of data | No |
| datadom-type | • dom<br>• xs:data<br>• xs:datatime<br>• xs:string | XML type of data | No |
| sql-type | • clob<br>• date<br>• xmltype<br>• varchar | SQL Type of data | No |

## Examples

When ancestor is ***fm:storage-location-list***

This will create a unique value, only used as part of ***fm:cache-key***

```
<fm:using using-type="UNIQUE"/>
```

Without defining using-type, it is assumed contents of ***fm:using*** are of using-type XPATH

```
<fm:using>:{params}/ID</fm:using>
```

When ancestor is ***fm:db-interface-list***

datadom-location is not needed when ***fm:using*** contains a child text node

```
<fm:using name=":bind">:{root}/DOCUMENT</fm:using>
```

Is the same as

```
<fm:using name=":bind" datadom-location=":{root}/DOCUMENT"/>
```

Fox will try and best guess the sql-type and datadom-type that the datadom-location is pointing too. Most text nodes will be set to xs:string and varchar respectively, however it's usually a good idea to explicitly set types of date elements as xs:date/xs:datetime and date. If trying to use XML as a type then XMLType and dom are typically used.

```
  <fm:using name=":bind" sql-type="varchar" datadom-type="xs:string">:{root}/TEXT_CONTENT/text()</fm:using>
  <fm:using name=":bind" sql-type="date" datadom-type="xs:date">:{root}/DATE_CONTENT</fm:using>
  <fm:using name=":bind" sql-type="xmltype" datadom-type="dom">:{root}/XML_CONTENT</fm:using>
```

### Related

- fm:cache-key
- fm:query
- fm:insert
- fm:update
- fm:lock
- fm:api

# FOX:Module:version-desc

### Description

***fm:version-desc*** is a child of ***fm:header***. It is used to store description of the current revision of the module, which is usually filled in via the developer's revision comment in the version control software.

### Syntax

```
<fm:header>
  <fm:version-desc>[ Version Description ]</fm:version-desc>
  ...
</fm:header>
```

### Examples

```
<fm:header>
  <fm:version-desc>Query updated to sort result list by ASC</fm:version-desc>
</fm:header>
```

### Related

- fm:header

# FOX:Module:version-no

## Description

***fm:version-no*** is a child of ***fm:header***. It is used to store a version number for the current revision of the module, which is usually filled in automatically through version control software.

## Syntax

```
<fm:header>
  <fm:version-no>[ Version Number ]</fm:version-no>
  ...
</fm:header>
```

## Examples

```
<fm:header>
  <fm:version-no>1.34</fm:version-no>
</fm:header>
```

## Related

- fm:header

# FOX:Module:view-rule

## Description

<fm:view-rule> is used within a <fm:security-list> and can be used to ENABLE a namespace in **read-only** mode. The namespace, privilege, state and theme attributes can all contain comma separated conditions.

This provides a way of overriding active namespaces for set-out/menu-out. It is a further final way of filtering the content of the screen under different conditions.

## Attribute Summary

| Attribute | Data Type | Description | Required |
|---|---|---|---|
| namespace | Literal String | The namespace for which to enable/disable read only mode. Can be a comma separated list. | Yes |
| operation | Literal String | ENABLE or DISABLE the read only rule. | Yes |
| privilege | Literal String | The privilege(s) the user must have to view the namespace. Can be a comma separated list. | No |
| state | Literal String | The sate(s) the user must be in to view the namespace. Can be a comma separated list. | No |
| theme | Literal String | The entry theme(s) the user must be in to view the namespace. Can be a comma separated list. | No |
| xpath | XPath String | The XPath test run relative to the root element. If it evaluates to true, the rule is applied. | No |
| rule-ref | Literal String | The name of a security-rule whose conditions to implement. | No |

## Examples

Generic:

```
<fm:security-list>
 <fm:view-rule namespace="namespace" operation="ENABLE" [privilege="user privilege" state="state" theme="entry-theme" xpath="XPath" rule-ref="rule-name"]/>
</fm:security-list>
```

A view-rule which only allows a user to view the 'department' namespace when they are in the 'view' entry theme and hold the privilege 'DEPARTMENT':

```
<fm:security-list>
 <fm:view-rule namespace="department" operation="ENABLE" privilege="DEPARTMENT" state="state" theme="view"/>
</fm:security-list>
```

## Related

- <fm:mode-rule>
- <fm:security-list>
- <fm:security-rule>

# FOX:Module:xml-commit

## Description

***fm:transaction-procedural-state*** is now deprecated and ignored by the FOX Engine.

## Related

- fm:control