MMBase Editwizard Reference Manual

Kars Veling, Q42
Henk Hangyi, MMatch
Pierre van Rooden, Publieke Omroep

Version 1.0 is released with MMBase 1.6.

Table of Contents

Introduction

This reference describes the syntax of the wizards and how to access them using a browser.

Calling the Wizard Pages

This reference describes the wizard pages and how to call them from a html or jsp page.

General - wizard definition schemas

The editwizards need a xml file with a definition of the wizard to operate. By default, those are found in the subdirectory /editwizard/data/, but it is possible to place your definitions, as well as the wizard stylesheets (which define layout) elsewhere.

When the wizards search for a requested file (a wizard xsl stylesheet or a wizard xml schema), they look for the referenced file in the following order:

The referrer page directory	You call the editwizard scripts (list.jsp and wizard.jsp) from a page located in another directory than the editwizard home directory. If you do, the wizards keep a reference to the calling page, the 'referrer'. When a xml or xsl file is requested, the wizard first checks whether the file can be found in the directory of the referrer page. Example: if the file requested is data/my_wizard.xml, and the referencing page is /myeditors/index.jsp, the system first checks for the file /myeditors/data/my_wizard.xml.
The editwizard home directory	The editwizard home is the directory that contains the editwizard basic stylesheets, data library, and jsp pages. It is one directory lower than the location of the list.jsp and wizard.jsp files. In the standard distribution, this location is /mmapps/editwizard/ (the list.jsp and wizard.jsp are located in /mmapps/editwizard/jsp/), but you can place the wizards wherever you like. Example: if the file requested is data/my_wizard.xml, and the editwizard home is in mmapps/editwizard/, the system checks for the file /mmapps/editwizard/data/my_wizard.xml.

list.jsp

To run a wizard, you'll need to call either list.jsp or wizard.jsp. These scripts are located in the jsp subdirectory of the editwizard home directory. In the standard distribution, this is mmapps/editwizard/jsp/.

List.jsp starts with running a search query on MMBase to create a list of items to search from. It shows a list of all found nodes and allows you to select one of these items to edit, to remove an item, or to add a new item. The actual options given depend on the security settings (if you are allowed to do this by MMBase), and the possibilities offered by the wizard. Note that the list.jsp does NOT run an editwizard itself, it is just a 'starter'. It eventually calls wizard.jsp (see below) to actually edit or create an item.

You need to specify a number of parameters to tell the list.jsp what wizard should be used, and what nodes and fields should be shown in the list.

In addition, by specifying the searchfields parameter, you can let the wizards generate a searchbox, allowing you to narrow the list of nodes to edit.

The following parameters MUST be specified (list.jsp does not work if you do not specify them):

wizard	The wizard to use. This is a relative reference to the xml schema file. Do not specify the ".xml" prefix. `wizard=data/simple.xml`

nodepath The nodepath defines the object types to list. This can be one object type, or a comma-separated list of types, which are treated as a relation chain, like in path attribute of the taglib's <mm:list> tag. The last objecttype listed is the object that is used as the objecttype to edit in the wizard.

```
nodepath=people or nodepath=people,news
```

The following parameters MAY be specified:

fields The fields to show in the list. If you query only one nodetype (see nodepath), you can use simple field names. Otherwise, you need to preface the fieldnames with the name of the objecttype they belong to. If the first field listed is a number field, the objecttype of that field is used as the objecttype to edit in the wizard (instead of the last object type in the nodepath list). This parameter is similar to the fields attribute of the taglib's <mm:list> tag.

Note

In 1.6.3 and lower, field is mandatory. In 1.6.4 and up, if fields is not specified, the wizard uses the default 'list' fields specified by MMBase.

```
fields=news.title,people.firstname,people.lastname
```

templates The directory where the xsl templates can be found. Use this to specify an alternate directory for xsl templates.

```
templates=data/my_xsl
```

main The name of the main node type in a node path. The main node type designates the element in a list that you want to edit. By default, the wizard takes the last name in a node path as the main node type.

For instance, in the node path 'mags,news,publishtimes', the main node type recognized automatically will be 'publishtimes' - but it is more likely that we desire to edit the news element in this list. In this case, you can point out to the wizard to use the 'news' element of the list instead.

Note

Available from MMBase 1.6.4 and up. Prior to 1.6.4, you can designate the main node type by adding the number field of that node type to the fields parameter (i.e: 'news.number,news.title,publishtimes.date'). This method still works in 1.6.4, but is discouraged as it is confusing.

```
main=news
```

sessionkey A key that identifies the 'session' of this wizard. Normally, you cannot run two separate editwizards concurrently from the same browser. If you want to allow this for your users for some reason, you can provide a 'sessionkey' to identify separate sessions: a wizard with one sessionkey runs independent from those who use a different one. The default session key is 'editwizard'. Note that allowing more wizards to be opened from one browser consumes more resources and is therefor not recommended.

```
sessionkey=my_key
```

context

The security context to assign to new objects and relations created by the editwizard. By default, objects are created using a default context defined by the security system (often depending on the logged on user). By explicitly specifying this default context. Not that specifying a context different from the default may result in objects that, once created, cannot be changed by the user due to security restrictions. A user needs to be able to change the context on objects he creates, or the parameter cause security errors.

```
context=default
```

maxsize

The maximum size of a file to upload. The default maximum size for files is 4MB.

```
maxsize=8000000
```

orderby

A list of fieldnames used to order the returning list. This parameter is similar to the orderby attribute of the taglib's <mm:list> tag.

```
orderby=people.lastname,news.title
```

directions

A list of one or more UP/DOWN keywords. Each keyword determines whether the order of a matching field specified in the orderby parameter is ascending (UP) or descending (DOWN). This parameter is similar to the directions attribute of the taglib's <mm:list> tag.

```
directions=UP
```

searchdirs

The direction in which relations between nodemanagers should be followed. This parameter is similar to the searchdirs attribute of the taglib's <mm:listcontainer> tag. It can be a list, in which case the first is for use between the first and second nodemanager, the second is for use between second and third nodemanager. If there are more steps in the nodemanager path then searchdirs, then the searchdir will default to the previous searchdir in the 'searchdirs' list.

Note

Available from MMBase 1.7.0 and up. In MMBase 1.6.4 and up, only the 'searchdir' parameter was available and only one searchdir could be specified, valid for all steps of the nodepath.

```
searchdirs=source,destination
```

constraints

A SQL-like constraint, used to filter the list. If you use fieldnames in a constraint, surround it with brackets ([]). This allows MMBase to recognize the fields and optimize the query. This parameter is similar to the constraints attribute of the taglib's <mm:list> tag.

```
constraints=[news.title] like '%Brasil%'
```

distinct

If true, double entries in a list are returned only once.

```
distinct=true
```

language

The preferred language for display. The language is specified by the ISO standardized 2-letter lowercase languagecode. The effects are dependent on whether alterenate langauge elemenst are availabel for the specified lan-

	guage. If not, the default language settings are used instead. `language=nl`
pagelength	The maximum number of entries shown on one list page. Default value is 50. `pagelength=50`
maxpagecount	The maximum number of pages available to browse through. Default value is 10. `maxpagecount=10`
start	The number of the item in the list where to start listing. This value is used to skip to a certain page when browsing a large list. Default is 0. `start=0`
age	The maximum age (in days) of the objects to list. Default is no age restriction. `age=31`
searchfields	The field(s) on which to make a search, using the value specified in the search parameter and the type of comparison specified in the search type. You can specify more than one field (separated by commas). if you do, a search matches if one or more of the specified fields matches.

Note

If the search parameter (see below) is set to AUTO, specifying this parameter will create a searchbox at the top of the list, allowing you to enter the value to search for. If a search box is forced by the search parameter but the searchfields are not specified, the default is to search on the fields being displayed (specified in the fields parameter)

```
searchfields=news.title,news.subtitle
```

realsearchfield	The actual field on which to search. Normally, searches are done on the fields mentioned in the searchfields parameter (used to create the search box in a list page). Sometimes, you want to search on a field other than those listed in searchfield (such as 'number' or 'owner'). In that case, use this parameter to specify the field.

Note

This parameter is set by the system when a user submits a search- you will hardly ever use it yourself. If you want to add additional constraints, use the 'constraints' parameter. If you specify a constraints parameter, that constraint is combined in further searches with the search parameters given by a user. If you use realsearchfield to specify a constraint, that information is only valid for the first search - it will be replaced when a user submits his search.

Note

Available from MMBase 1.6.4 and up

```
realsearchfield=news.title,news.subtitle
```

searchvalue

The term or value to search for. This is only valid if you also specify the searchfields parameter.

```
searchvalue=Brasil
```

searchtype

Determines what type of search is used to filter on the searchfields, using the specified search value. This is only valid if you also specify the searchfields parameter.

Allowed values are 'string', 'like', 'equalto', 'greaterthan', 'lessthan', 'notgreaterthan', and 'notlessthan'. use 'string' to compare the search value to the field using an exact match. Use 'like' to make a pattern search using the specified searchvalue. Use the other values to compare the field to numeric values or dates. I.e. using 'greaterthan' selects all fields whose (numeric) value is greater than the specified (numeric) value.

```
searchtype=like
```

search

The search parameter specified when the list should show a search box, and whether searching is mandatory. There are four possible values:

no turns searching off. If other parameters such as searchfields and searchvalue are given, the search is performed using those parameters but a searchbox is not displayed.

yes turns searching on, displaying a searchbox on top of the list, using defaults for parameters such as searchfields, if these are not specified.

force is like YES, but searching (and showing a resultlist) is not performed until a searchvalue is specified. If no searchvalue is given, only the searchbox is displayed - no results are shown.

auto makes searching dependent on the presence of the searchfields parameter. If the searchfields parameter is given, a searchbox is shown. Otherwise, it is not. AUTO is the default value

Note

Available from MMBase 1.6.4 and up

```
search=yes
```

origin

A node alias or number, which will be passed to the underlying editwizard. the value can then be referenced in the editwizard schema using the {$origin} variable. This can be used to create relations to nodes whose 'origin' is determined outside the wizard (i.e. in a complex search or determined through user preferences).

```
origin=my_start_node
```

title

An optional title to use by the list. In general, you should specify this in the editwizard schema.

```
title=News of The Week
```

proceed	System parameter, see the description at the wizard.jsp reference page for more info
replace	System parameter, see the description at the wizard.jsp reference page for more info
remove	System parameter, see the description at the wizard.jsp reference page for more info

Example 1. How to use list.jsp to list objects

```
list.jsp?wizard=practice/people&nodepath=people&fields=firstname,lastname
```

```
list.jsp?wizard=practice/simple&startnodes=1&nodepath=people,news&fields=news.t
```

wizard.jsp

You can use wizard.jsp to directly start a wizard. Like list.jsp, you will need to supply the correct parameters in order to let the wizard run correctly. If you call the wizard.jsp page, a wizard will be loaded and started. The proper html will be rendered and shown in the browser. The syntax of wizard.jsp to create a new object looks like this:

```
wizard.jsp?wizard=practice/simple&objectnumber=new
```

To create a wizard and use an existing mmbase object as source:

```
wizard.jsp?wizard=practice/simple&objectnumber=[objectnumber]
```

The following parameters MUST be specified (wizard.jsp does not work if you do not specify them):

wizard	The wizard to use. This is a relative reference to the xml schema file. Do not specify the ".xml" prefix. `wizard=data/simple.xml`
objectnumber	The number or alias of the object to edit. The object must exist. You can also supply the keyword 'new', which creates a new object instead of editing an existing one.

The following parameters MAY be specified:

templates	The directory where the xsl templates can be found. Use this to specify an alternate directory for xsl templates. `templates=data/my_xsl`
sessionkey	A key that identifies the 'session' of this wizard. Normally, you cannot run two separate editwizards concurrently from the same browser. If you want to allow this

for your users for some reason, you can provide a 'sessionkey' to identify separate sessions: a wizard with one sessionkey runs independent from those who use a different one. The default session key is 'editwizard'. Note that allowing more wizards to be opened from one browser consumes more resources and is therefor not recommended.

```
sessionkey=my_key
```

context
: The security context to assign to new objects and relations created by the editwizard. By default, objects are created using a default context defined by the security system (often depending on the logged on user). By explicitly specifying this default context. Not that specifying a context different from the default may result in objects that, once created, cannot be changed by the user due to security restrictions. A user needs to be able to change the context on objects he creates, or the parameter cause security errors.

```
context=default
```

language
: The preferred language for display. The language is specified by the ISO standardized 2-letter lowercase languagecode. The effects are dependent on whether alterenate langauge elemenst are availabel for the specified language. If not, the default language settings are used instead.

```
language=nl
```

debug
: If debug=true, the wizard will place additional information in the wizard page that allows for debugging of the application. This includes a link to a debug page that allows you to view the loaded object data and the wizard schema. This link specifies parameters such as sessionkey and popupid as required.

```
debug=true
```

maxsize
: The maximum size of a file to upload. The default maximum size for files is 4MB.

```
maxsize=8000000
```

origin
: A node alias or number. This value can be referenced in the editwizard schema using the {$origin} variable. This can be used to create relations to nodes whose 'origin' is determined outside the wizard (i.e. in a complex search or determined through user preferences).

```
origin=my_start_node
```

popupid
: This parameter is passed by the system, you never pass it yourself. It is used to identify a 'popup' wizard. A popup wizard is a sub-wizard for the main wizard that, unlike an 'inline' sub-wizard, opens in a separate browser window. Because of this, it needs to separately maintain its status (identified by the popupid). Once the task of a popup wizard ends, it's window is closed, and the parent wizard is refreshed.

Note

You never pass this parameter by hand. They are provided by the <command name="startwizard"> and <field ftype="startwizard"> tags in a wizard schema. However, you might need to pass the parameter in links if you override xsl stylesheets, and if you want to use the debug.jsp page to view a popup wizard's debug data, you need to know the value of this parameter.

```
popupid=my_id
```

proceed This is a parameter passed by the system. It is used to indicate that the current wizard is still being processed, and therefor its data (stored in the user's session) should be kept an re-used by this call. Omitting this parameter (or passing it with the value false) causes all processed data to be removed, and replaced by a new wizard.

Note

You never pass this parameter by hand, but you might need to pass it in links in overriding xsl stylesheets.

```
popupid=my_id
```

replace System parameter

Note

You never pass this parameter by hand. It is used to by the wizard to pass state information.

remove System parameter

Note

You never pass this parameter by hand. It is used to by the wizard to pass state information.

Example 2. How to use wizard.jsp to start an editwizard

```
wizard.jsp?wizard=practice/simple&objectnumber=184
```

```
wizard.jsp?wizard=practice/simple&objectnumber=new
```

debug.jsp

When you are debugging your wizard code, it can be useful to know what data is read and parsed by the wizard. The debug.jsp page gives access to three xml trees that are used by the current running wizard.

The data tree shows the xml that contains the data used by the wizard, including any changes made by the user. It can be useful to see if data is loaded (or changed) as expected. It can also be useful to determine xpaths when you plan to use these in your wizard: if you use an xpath in your wizard xml file, it access the data xml.

The wizard lists the fully expanded wizard schema file (including any inclusions and extensions). Note that this xml also contains some additional navigational information related to the current form, and does not fully follow the wizard dtd.

The wizard-schema combines the data and wizard xmlks into one xml tree. This tree is what is passed to the xsl-transformers, and is used to create the form that is sent to the end-user. Use this form to see if data loaded is actually used, and if you want to override the default xsl files.

The following parameters MAY be specified:

sessionkey A key that identifies the 'session' of this wizard. See the wizard.jsp for more information.

	`sessionkey=my_key`
popupid	Specify this parameter to view a popup wizard's debug data (instead of the data displayed by the main wizard). See the popupid of the wizard.jsp for more info. `popupid=my_id`

Basic Concepts

TODO

Object structures (Dove)

TODO

XPath Expressions and Wizard variables

TODO

Including xml fragments

TODO

Extending using xml fragments

TODO

Syntax Reference

{$origin}

Usage. The $origin variable contains the value of the 'origin' parameter passed to the wizard.jsp or list.jsp pages.

The variable can be used as an expression in the object-structure of the add action and create action tags, and as a parameter value in the search command and wizard command tags. This allows you to create objects that are automatically linked to the passed owner object, or to pass owner information to another wizard or list, allowing for more generic wizards.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        <action type="create" >
          <object type="news">
                  <field name="title">My Title</field>
                  <relation role="related" destination="{$origin}" />
                </object>
        </action>

  ...

</wizard-schema>
```

{$pos}

Usage. The $pos variable identifies the index of an object in a list of objects returned by the search command.

The list of objects is inserted at the end of the list that called the search command. Some relations have fields whose values are initialized by the wizard a they are inserted. In a few cases, the value inserted may be calculated based on values of the current list. For instance, the pos field of a posrel relation will often need to be filled with a sequentially increasing number. This can be done by including an xpath in the field tag of the relation's create action. For the posrel field, this can be:

```
<action type="create">
        <relation role="posrel" type="news">
                <field name="pos">{sum(//relation[@role='posrel' and @lastitem=
        </relation>
</action
```

Which gets the highest number in the original list, and increases it by 1. (Note how we use 'sum' so we get a value of 0 if there are no items in the list)

However, this code will not always do what we want: if a search result returns more than one object, all the pos fields are assigned the same number (as the xpath runs on the original data).

In that case, you can make use of the $pos variable. The $pos variable, on creating a relation to a searched object, contains the index of that object in the resultlist, starting with 0 for the first item in the list. Including the $pos variable in the xpath gives:

```
<action type="create">
        <relation role="posrel" type="news">
                <field name="pos">{sum(//relation[@role='posrel' and @lastitem=
        </relation>
</action
```

This creates a value that is equal to the highest number in the original list, increased by the object's index in the list + 1. The result is that the 'pos' fields of each new relation are guaranteed to be uniquely numbered (and all higher than the original highest number).

Note that this manner of initializing fields only really works when combining it with sorted lists.

action

Syntax.

```
<action [type="add|create|delete|load" />
```

Usage. The action object defines whether, and what object structures the wizard is to create, load, or delete when the appropriate command is given. Each action is implicitly associated with a certain command. The top-level actions are associated with commands issued from outside, or on startup of the wizard (such as loading the correct objectstructure when you start a wizard to edit data, or a command to delete a certain object from a list).

Actions in lists define what object structures to create or delete when operations are issued on a list's contents. The content of the action is generally an objectstructure (with the exception of the top-level delete action), which defines the data that is to be handled. More info can be found in the reference for the individual action types.

Required Attributes.

type	The type of action. Either add, create, delete, or load. For more info see action type="create", action type="add", action type="delete" and action type="load"

Optional Attributes. None

Required Child Nodes. None

Optional Child Nodes. See the individual types of actions for more details

Possible Parents.

<wizard-schema>	The wizard schema root (for top-level actions).
<list>	A list in a form (for list mutation actions).

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        <action type="create" >
          <object type="news">
                 <field name="title">My Title</field>
                </object>
        </action>

        <form-schema id="step1">
                <field name="title">
                        <prompt>Title</prompt>
                </field>
        </form-schema>
</wizard-schema>
```

action type="add"

Syntax.

```
<action type="add" />
```

Usage. This tag defines how to create a new relation in a list, to an object supplied by one of the list commands. It is similar to action type="create", but it is used for the command name="search" or command name="wizard" tags.

Under the tag, you specify a relation tag, with the role and destinationtype attributes set to the type and role of the relation to create:

```
<action type="add" >
        <relation role="author" destinationtype="people" />
</action>
```

This action creates a 'author' relation to a people object. The wizard expects that the object is supplied by one of the list command tags (a search command or a wizard command).

See the action type="create" for information on how to create a new related object in the list.

Required Attributes.

type	The type of action, in this case, "add".

Optional Attributes. None

Required Child Nodes.

<relation />	Specifies the type of relation to create in a list. It may also include additional field and object tags for initialization of the object or creation of new objects to relate to.

Optional Child Nodes. None

Possible Parents.

<list>	A list of related objects in a form.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        ...

        <form-schema id="step1">
                <field name="title">
                        <prompt>Title</prompt>
                </field>

                <list role="author" destinationtype="people" >

                        ...

                        <command name="search" ... >

                        <action type="add" >
                                <relation role="author" destinationtype="people
                        </action>
                </list>
        </form-schema>

</wizard-schema>
```

action type="create"

Syntax.

```
<action type="create" />
```

Usage. This tag defines how to create a new object or relation/object. As a top-level tag it defines how to create and initialize a new main object. In a list, it specifies how to create a relation to a newly instantiated (empty) object in that list. Not specifying a create action means the action is not available (i.e. you cannot create a main object in a wizard or a new related object in a list)

You define the data to be created by including a (nested) objectstructure in the body of the action tag.

Under a top-level tag, you specify an object tag, with the type attribute set to the type of object that needs to be created.

```
<action type="create" >
```

```
        <object type="news" />
</action>
```

This action creates an (empty, and unrelated) news object.

You can make a more complex creation action by specifying fields that you want to initialize (aside from any initialization that is done by MMBase itself), or relations that you want to have made (such as linking to a specific archive):

```
<action type="create" >
        <object type="news" >
                <field name="title">My Title</field>
                <relation role="related" destination="my_news_archive" />
        </object>
</action>
```

This will initialize the 'title' field of the news object, and relate it to an object (i.e. a pool object) with the alias 'my_news_archive'. There are a limited number of 'variables' that you can use when initializing the action. More info on this can be found in the relation and field tag references.

As a list-level tag, the action is similar to action type="add", but it is used for the command type="insert" tag.

You specify a relation tag, with the role and destinationtype attributes set to the type and role of the relation to create. You can also specifying relation fields that you want to initialize. You also need to add an 'object' tag (with initialized fields or even additional relations to create), which prompts the wizard to create the object:

```
<action type="create" >
        <relation role="author" />
                <object type="people" >
                        <field name="lastname">Supply lastname here</field>
                </object>
        </relation>
</action>
```

This will create a new 'people' object with an initialized 'lastname' field, and create a relation to this object. There are a limited number of 'variables' that you can use when initializing the action. More info on this can be found in the field tag reference.

You use this action to define how to create a relation for a list insert command. If you have a search or a wizard command tag, you should use an action type="add" tag instead.

Note

You can use both the add and create actions in a list, allowing you to supply both a search and a insert command.

Required Attributes.

type	The type of action, in this case, "create".

Optional Attributes. None

Required Child Nodes.

<object /> (top-level tag)	Specifies the type of object to create. It may also include additional field and relation tags for initialization of the object or auto-linking to other objects.

(list-level tag)	Specifies the type of relation to create in a list. It may also include additional field and object tags for initialization of the object or creation of new objects to relate to.

Optional Child Nodes. None

Possible Parents.

<wizard-schema> (top-level tag)	The wizard schema root.
<list> (list-level tag)	A list of related objects in a form.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        ...

        <action type="create" >
          <object type="news" >
                <field name="title">My Title</field>
                      <relation role="related" destination="my_news_archive" 
                </object>
  </action>

        <form-schema id="step1">
                <field name="title">
                      <prompt>Title</prompt>
                </field>

                <list role="author" destinationtype="people" >

                      ...

                      <command type="insert" />

                      <action type="create" >
                              <relation role="author" destinationtype="people
                                <object>
                                        <field name="lastname">Insert name he
                                      </object>
                              </relation>
                      </action>
                </list>
        </form-schema>

</wizard-schema>
```

action type="delete"

Syntax.

```
<action type="delete" />
```

Usage. This tag defines actions to take when an object is to be removed. there are two possible occurrences of this action: as a top level tag, or as a list-tag.

A top level tag defines an action for when a user issues a delete command from the wizard list page. If you define a delete action, list pages will enable the user to remove an object from the MMBase cloud by showing a delete button or link. Not defining this action means the link does not show (thus, objects cannot be deleted if you do not provide the action). Optional children of the action are a roll-over help-description for the link in the list, and a prompt for asking confirmation when the command is issued.

Note

Unlike other actions, the top level delete action does not contain an objectstructure - the object deleted is always the current object and its relations. You cannot, for example, define a structure so that deleting the node will automatically delete any related nodes.

A list tag defines an action for when a relation is removed from a list in a form. In general, you need not supply this action - deletes are generally handled automatically. Delete buttons in forms are enabled or disabled by the list tag, not by the delete action. The only time you want to use the delete action in a list is when you want to remove both a relation and the related node. In this case, you can add the delete action, along with the 'additional' objectstructure to delete (besides the relation itself). Currently this 'objectstructure' is only the underlying object, so a delete action will generally look like:

```
<action type="delete">
  <object />
</action>
```

Required Attributes.

type	The type of action, in this case, "delete".

Optional Attributes. None

Required Child Nodes. None

Optional Child Nodes.

<description /> (top-level tag)	Descriptive text used as help text (often a rollover) for the 'delete' link in the wizard list. Only applicable to the top-level delete action.
<object /> (list-level tag)	If given, the delete action not only removes the relation, but also the related node. This can be useful if you have one-use only objects, which you like to have cleared when they are unlinked from their 'parent' object. Only applicable to the list-level delete action. Note that unlike other actions, the object child tag cannot itself contain other relations or objects.
<prompt /> (top-level tag)	Text to prompt when asking confirmation when the command is issued. This allows you to give specific warnings. Only applicable to the top-level delete action.

Possible Parents.

<wizard-schema>	The wizard schema root (for top-level delete actions).
<list>	A list in a form (for list delete actions).

Example.

```
<wizard-schema id="my_wizard">
```

```
        <title>Wizard-Title</title>

        ...

        <action type="delete" >
          <prompt>Do you really want to delete this item?</prompt>
                <description>Click to delete this item</description>
        </action>

        <form-schema id="step1">
                <field name="title">
                        <prompt>Title</prompt>
                </field>

                <list role="related" destinationtype="sections" >

    ....

                <action type="delete">
                  <object />
                </action>

                </list

        </form-schema>
</wizard-schema>
```

action type="load"

Syntax.

```
<action type="load" />
```

Usage. This tag defines actions to take when an object is to be loaded by the wizard for editing. It is a top-level tag and always applies to the main edit object (the object whose number is passed through the objectnumber parameter when calling wizard.jsp).

By defining a load action, you can command the wizard to load specific fields, and additional (related) objects to be edited. You define the data to be loaded by including a (nested) objectstructure in the body of the action tag. Directly under that tag, you can specify the fields you want to load (using field tags), and the relations from his node that you want to follow and load the data of. Loading a relation also loads the related node. As an example:

```
<action type="load" >
        <field name="title" />
        <field name="subtitle" />
        <relation destinationtype="images" role="posrel" />
</action>
```

This action loads an object's title and subtitle fields, as well as all images that are related to the object through the 'posrel' role.

This action loads all the fields of the 'posrel' relation, and those of the related images. If you want to limit the fields that are loaded (for instance to limit the amount of space the wizard uses), you can specify this by expanding the objectstructure as follows:

```
<action type="load" >
        <field name="title" />
        <field name="subtitle" />
        <relation destinationtype="images" role="posrel" >
                <field name="pos" />
                <object>
```

```
                        <field name="title" />
                </object
        </relation>
</action>
```

This will load the 'pos' field of the posrel relation, and the 'title' field of any related images. You can also continue to nest relations, such as the image related to an author related to a news item. This makes for more complex wizards, so in general you will not go more than one relation deep.

If you do not specify a load action, the wizard by default loads the main object with all its (editable) fields. It will in that case not load any related objects.

Note

You can only edit fields or list related objects if the wizard has loaded them. If you specify too few fields in your load action, you may not be able to access everything you want to edit. Likewise, specifying too much may make the wizard slow.

Required Attributes.

type	The type of action, in this case, "load".

Optional Attributes. None

Required Child Nodes. None

Optional Child Nodes.

<field />	If given, the load action only loads those fields for which there are tags defined. If you do not list any field tags, all fields are loaded.
<relation />	If given, the load action loads the relation as specified, as well as the objects related to. Relations can contain field and object tags, which can further determine what is loaded by the wizard.

Possible Parents.

<wizard-schema>	The wizard schema root.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        ...

        <action type="load" >
          <field name="title" />
          <field name="subtitle" />
          <field name="intro" />
          <field name="body" />
                <relation destinationtype="images" role="posrel"  >
                  <field name="pos" />
                        <object>
                                <field name="title" />
                        </object
                </relation>
                <relation destinationtype="attachments" role="related" >
                <relation destinationtype="urls" role="related" />
                <relation destinationtype="people" role="author" >
                        <object>
                                <field name="title" />
                                <relation destinationtype="images" role="relate
                                        <object>
                                                <field name="title" />
                                        </object
                                </relation>
                        </object
                </relation>
  </action>

        <form-schema id="step1">
                <field name="title">
                        <prompt>Title</prompt>
                </field>
        </form-schema>
</wizard-schema>
```

form-schema

Syntax.

```
<form-schema [ id="{form_schema_id}" ] />
```

Usage. A wizard will always be needing at least one form-schema. In a form-schema one form of a wizard is defined. A form typically contains fields to edit, or lists of relations to change. Each form-schema is represented as one (web) page in the wizard.

Required Attributes. None

Optional Attributes.

id	The identifier of the form. Use this id to refer to the form from, for instance, the step tag.

Required Child Nodes. None

Optional Child Nodes.

<title />	The title of a form
subtitle	The subtitle of a form
<field />	The presentation of an object field in the form. You can define more than one field.
<fieldset />	A set of fields to present in the form. You can define more than one fieldset.
list	The presentation of a list of relations in the form. You can define more than one list.

Possible Parents.

<wizard-schema>	The wizard schema root.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        <form-schema id="step1">
          <title>Step 1</title>
          <subtitle>Enter basic data</subtitle>

                <field name="firstname">
                        <prompt>First Name</prompt>
                </field>
                <field name="lastname">
                        <prompt>Last name</prompt>
                </field>

          <list ...>
                 ...
    </list>

        </form-schema>

</wizard-schema>
```

item

Syntax.

```
<item [displaytype="image"] />
```

Usage. An item describes how and what to display for each element in a list.

An item is similar to a form-schema, but it's content is displayed in a list body, and possibly provided with buttons for removing the item from a list or moving it up and down if an order was specified.

The fields within the <item> tag assume that the object that is to be displayed or edited is the related object from a list (not the relation). So in a list of 'news' objects accessed through the 'posrel' role, the field with name "title" displays the title of the related news object.

If you want to display a field from the relation you need to specify a fdatapath instead. i.e. to display the "pos" field of a posrel relation:

```
  <field fdatapath="field[@name='pos']" ftype="data" />
```

You can also specify a field from the related object with a fdatapath, like this:

```
  <field fdatapath="object/field[@name='title']" />
```

Which is the same as:

```
  <field name="title" />
```

Required Attributes. None

Optional Attributes.

displaytype	The displaytype attribute can be specified if you desire the item to be shown in a special way. The stylesheet used to craete wizard pages (wizard.xsl) should be altered so it recognizes the displaytype (otherwise it will be ignored). The default stylesheet provides the displaytype "image" which assumes the object to be shown is an image object. It adds a thumbnail of the image to the item and sdiplays any addiitonal field next to it.

Required Child Nodes. None

Optional Child Nodes.

<title />	The title of the item. Normally ignored.
<description />	The description of the item. Normally ignored.
<field />	The presentation of an object field in the item. You can define more than one field.
<fieldset />	A set of fields to present in the item. You can define more than one fieldset.
list	The presentation of a list of relations in the item, related to the related objectd isplayd in the item. You can define more than one list. Note that lists in an item may show awkard depending on the wizard stylesheet used.

Possible Parents.

list	The list providing the data.

Example.

```
<list role="related" destination="categories" orderby="name" hidecommand="move-

  <title>Category</title>
  <item>
    <field name="name" ftype="data" />
    <field name="description" ftype="data" />
   </item>

  <command name="search" nodepath="categories" fields="name,description" orderb
    <search-filter>
      <name>Name contains</name>
      <search-fields>name</search-fields>
    </search-filter>
  </command>

  <action type="create">
    <relation role="related" destinationtype="categories" />
  </action>

</list>
```

list

Syntax.

```
<list [role="{role}"] [destinationtype="{objecttype}"] [searchdir="source|desti
        [fdatapath="{xpath}"] [fparentdatapath="{xpath}"]
        [ordertype="string|number"] [orderby="{xpath|fieldname}"]
        [minoccurs="{number}"] [maxoccurs="{number}"]
        [hidecommand="{commandlist}"] />
```

Usage. A list displays a set of related nodes, and provides means to edit this list - either by adding items to or removing items from the list, or by editing items directly. Lists can be sorted, and various commands can be added to allow a search for new objects to add, subwizards to call, or new empty objects to be created and inserted in the list.

Lists are similar to fields in that they select a certain element from the data document. In particular, List select relation nodes. The attributes of as list provide a filter with which the wizard selects relations, relative to the object that has focus (generally the main object). The filter can be set by specifying role of the relation or the destinationtype, but it is also possible to make a more complex filter, by providing an xpath using the fdatapath attribute.

An example of using role/destinationtype is the following:

```
<list role="posrel" destinationtype="news" >
        <item>
          <field name="title" ftype="data" />
        </item>
</list>
```

Which selects all 'posrel' relations to 'news' objects (seen from the current object), and shows the related objects title field. The list only shows those relations loaded or created by the wizard using the action tags.

You can also obtain the list noted above using the fdatapath attribute. In that case, you specify an xpath, relative from the current object, as follows:

```
<list fdatapath="relation[@role='posrel' and object/@type='news']" >
        <item>
          <field name="title" ftype="data" />
        </item>
</list>
```

See the section XPath Expressions and Wizard Variables for more info on using xpaths.

Displaying content is done by specifying an item tag inside the list. The item can contain field objects and other lists, with which you can display or edit the object's or the relation's fields.

Note

While the list 'selects' relations (in the example, the posrel relation), the fields within the <item> tag assume that the object that is to be displayed or edited is the related object (in the example, the news item). See the <item> tag for more info.

Required Attributes. None

Optional Attributes.

destinationtype	The destination type of a related object. This filter is combined with 'role' and 'searchdir' to select the objects to show in the list.

Note

In MMBase 1.6.3 or lower, this attribute is called 'destination'. While you can still use the 'destination' attribute in 1.6.4, you should consider using the 'destinationtype' attribute instead.

role

The role of the relation. This filter is used combined with 'destination' and 'searchdir' to select the objects to show in the list.

searchdir

The direction to follow the relation in. You can specify the values 'source', 'destination', or 'both'. 'Source' and 'destination' in this context refer to the directionality of the relation. Each relation in mmbase has a 'source' (where the relation starts) and a 'destination' (where the relation ends). Most of the time this directionality is not of import. You can follow relations either way without worrying on directionality, and most of the time only one relation is possible between two object types (so specifying destinationtype is often enough). In some cases you may have more possible relations (say, when you relate objects of the same type, i.e. a pool hierarchy). If you want to narrow down the selection based on direction of the relation, you specify searchdir.

If you specify 'source', only those relations are returned where the 'current' object is the destination object (and the related objects are the 'source'). Specifying 'destination' only returns those relations are returned where the 'current' object is the source object. 'Both' returns all relations.

This filter is used combined with 'role' and 'destination' to select the objects to show in the list.

fdatapath

The xpath describing the relations whose objects to show in this list. Note that if you specify fdatapath, you cannot use destination, role, or searchdir.

fparentdatapath

The xpath describing the parent object of the list. The value of this path is used when adding new relations (using the add or create actions), as the parent object of the list is the node new relations are added to.

Normally, the parent of a list is automatically determined. In general, it will be the object that has focus when the list is created - i.e. the main wizard object for a normal list, or an object identified by an 'item' tag, when using a list-within-a-list. Generally you do not have to specify this attribute.

However, if you specify a fdatapath attribute, it is possible to obtain a list of relations who are not the children of the focus object. In that case, if you want to be able to create new relations, you need to add the fparentdatapath to point out the parent object.

For example, suppose you use an 'employee' object, that contains contract data. It is related to one 'people' object (containing person data), which is connected to several 'contact' objects, containing alternate email addresses. If you want to edit the emailaddresses directly from the 'employee' object, you can dot so as follows:

```
<list fdatapath="relation/object[@type='people']/relation[objec
        <item>
          <field name="email" />
        </item>

        <command name="insert" />

        <action type="create">
          <relation role="related">
                  <object type="email" />
                </relation>
        </action>
</list>
```

This tag selects all email objects related to the people object related to the main (employee) object. By specifying the fparentdatapath, we can now add new email address, and they will be added to the correct object (that is, to the people object, rather than the employee object).

orderby

The field on which to order the list. By default, a list is unordered, and newly added items are added at the end of the list. By specifying a field you can order the list as you wish. The value of orderby can either be a fieldname of the related node, or a xpath..

For instance, using a fieldname:

```
<list role="posrel" destinationtype="email" orderby="title" hic
...
</list>
```

will sort the list on the title of the related email objects.

If the ordering need be on another property, such as the field of a relation or the number of a node, you specify an xpath Take into account that the xpath is relative to the relation selected by the list (and not to the related node). For instance:

```
<list role="posrel" destinationtype="email" ordertype="number"
...
</list>
```

will sort the list on the number of the related email objects, where as:

```
<list role="posrel" destinationtype="email" ordertype="number"
...
</list>
```

will sort the list on the number of the relations (the posrel).

Finally:

```
<list role="posrel" destinationtype="email" ordertype="number"
...
</list>
```

will sort the list on the number of the relations (the posrel).

When specifying a sort order, the wizard will automatically make up/down buttons that allow you to change the order of the objects in a list. The ordering is rather crude, as it merely switches the value of the 'orderby' field between objects. As such, when you order a list on a field you do not want to have changed (such as the title field of a newsitem or the number field, as in the examples above), you should disable these buttons using the hide-command attribute.

Also note that you if the order field is a numerical value, you will need to specify this using the ordertype attribute.

ordertype

The type of the orderby field. Possible values are string and number. This attribute determines whether sorting takes place on lexical or numerical value. I.e., with ordertype="string", the value '12' will come after '100'. With ordertype="number", the value '12' will come before '100'.

minoccurs	The minimum number of objects that should be in this list. If the list contains less than the indicated number, the wizard will disable the save button. The default minimum is 0 (no) objects.
maxoccurs	The maximum number of objects that are allowed in this list. If the list contains more than the indicated number, the wizard will disable the save button. This attribute can be a number or the character '*' (infinite). The default maximum is infinite.
hidecommand	A list of commands that should NOT be made available in the list. The attributes contains the command names, separated by a '|' character. The commands that can be turned off are move-up, move-down, delete-item, and add-item. By default, all commands are active when appropriate.

Note

The command can be used to turn off the move-up/move-down buttons when you want to sort the list using a fixed field.

Required Child Nodes. None

Optional Child Nodes.

<title />	The title of a list, often displayed as a caption or prompt
description	The description of a list, often shown as a help text in for instance a roll-over text
action	Defines how to manipulate a list when creating or deleting list items. Possible actions are add (add an existing object to the list) create (create a new object and add it), and delete (delete an object)
command	Defines ways to manipulate (add data) to the list. Possible commands are search (search for an object to include in the list), insert (create a new empty object and add it to the list), and startwizard (call a subwizard to create a new object to add to the list).
item	Item determines how a show a related object. It is similar to a form-schema, in that it can contain fields and lists.

Possible Parents.

<wizard-schema>	A representation of a form in the wizard.
item	An item in a (higher) list.

Example.

```
<list role="related" destination="categories" orderby="name" hidecommand="move-

        <title>Category</title>
        <item>
                <field name="name" ftype="data" />
                <field name="description" ftype="data" />
         </item>

        <command name="search" nodepath="categories" fields="name,description" 
                <search-filter>
                        <name>Name contains</name>
                        <search-fields>name</search-fields>
                </search-filter>
        </command>

        <action type="create">
                <relation role="related" destinationtype="categories" />
        </action>

</list>
```

object

Syntax.

```
<object [type="{objecttype}"] />
```

Usage. TODO

step

Syntax.

```
<step form-schema="{form-schema-id" />
```

Usage. A step in a wizard, defining a form to run. the position of a step in the parent steps tag determines the place of the form in the wizard.

Required Attributes.

form-schema	The id of a form-schema in this wizard.

Optional Attributes. None

Required Child Nodes. None

Optional Child Nodes. None

Possible Parents.

<steps>	Determines the order of forms in a wizard.

Example. See steps

steps

Syntax.

```
<steps />
```

Usage. A list of step tags, listing the order in which a wizard's forms need to be shown (one step is similar to one form). If you do not supply a steps tag, the wizard assumes that forms need to be shown in the order in which they are defined in the wizard. You generally do not need a steps tag.

Required Attributes. None

Optional Attributes. None

Required Child Nodes.

<step />	A step detailing a form to show. A steps tag should contain one or more step tags.

Optional Child Nodes. None

Possible Parents.

<wizard-schema>	The wizard schema root.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>

        <steps>
                <step form-schema="edit_news" />
                <step form-schema="add_info" />
                <step form-schema="publish" />
        </steps>

        <form-schema id="edit_news">
                <field name="firstname">
                        <prompt>Your Firstname</prompt>
                </field>
        </form-schema>

        <form-schema id="add_info">
          ...
        </form-schema>

        <form-schema id="publish">
          ...
        </form-schema>

</wizard-schema>
```

wizard-schema

Syntax.

```
<wizard-schema [ id="{wizard_schema_id}" ] />
```

Usage. This is the root-node of a wizard. Always use a wizard-node to start a wizard.

Required Attributes. None

Optional Attributes.

id	The identifier of the wizard. Not used at this time.

Required Child Nodes.

<title />	Title of the wizard.
<form-schema />	One or more forms for editing

Optional Child Nodes.

<action type="create" />	Instructions for creating a new node (by default you cannot create a node)
<action type="delete" />	Instructions for deleting a node (by default you cannot delete a node)
<action type="load" />	Instructions for loading data of an node to edit, including possible related nodes (bt default load a node and all its fields)
<description />	Description of the wizard.
<lists />	Define enumerated lists to use in the wizard.
<steps />	Orders form-schemas in a certain order (default order is the order of forms as defined in the wizard).
<task-description />	Description of the wizard. Obsolete. use <description />

Possible Parents. None, this is the root element.

Example.

```
<wizard-schema id="my_wizard">
        <title>Wizard-Title</title>
        <form-schema id="step1">
                <field name="firstname">
                        <prompt>Your Firstname</prompt>
                </field>
        </form-schema>
</wizard-schema>
```

Syntax reference

object (inside action type="create")

Syntax	<object type="[buildername]" />
Usage	Inside a create-action, you can place an object-node to define what should be created within mmbase if the create-action is performed. With this node, you can define what object should be created, what values should be placed in what fields, and you can define possible new relations that should be created.
Needed attributes	type="[buildername]"
Possible attributes	See needed attributes.
Needed childnodes	None
Possible childnodes	<relation /> If a relation node is placed inside an objectnode, a relation is created relating the object defined by the <object /> node and some other object (eg.: another to-be created object or an existing one.) <field /> In the field nodes inside the object nodes, the user can define the defaultvalues of the designated fields. Eg.: <field

	name="firstname">Enter your firstname</field>
Possible parents	<action type="create" />
Example	

```
<object type="news">
  <relation destination="13234" role="related" />
  <relation role="posrel">
    <field name="pos">42</field>
    <object type="images">
      <field name="title">new image</field>
    </object>
  </relation>
</object>
```

relation (inside action type="create")

Syntax	<relation />
Usage	Usually, the relation is placed inside an object-node (inside a create-action), or it will be placed directly inside a create-action. With this node, the user can make relations between two newly created objects, or, create a relation between one newly created object and an already existing object.
Needed attributes	None
Possible attributes	destinationtype="[buildername]" This attribute is not used anymore. Forget it I'd say.
	destination="[objectnumber or alias]" Use this attribute to point the relation to an already existing object in the mmbase cloud.
	role="[relationname]" Use this attribute to define what kind of relation should be created. If omitted, the default InsRel relation is used. Best practice is to always define the role, eg.: "related" or "posrel" etc.
Needed childnodes	None
Possible childnodes	<object /> (see also: object)
	<field /> Default values of fields of a relation itself can be set also. Eg.: <field name="pos">-1</field>
Possible parents	<action type="create" />
	<object />
Example	(See also: the <action type="create" /> example and the <object /> example.

field (general)

Syntax	<field />

Usage

Field nodes define what form-elements will be placed in the wizard. This fieldnodes in the xml are an important part of the wizard's definition. For detailed information on what fieldtypes need what kind of settings, see the other fielddefinitions in the reference. Note: the field nodes that can be placed inside a relation or object node have different syntax! (See also: <object type="create" />)

Needed attributes

name="[fieldname]" fdatapath, ftype, dttype, dtrequired, dtminlength, dtmaxlength are automatically retrieved from mmbase. Overriding is always possible, however.

or (advanced users:) fdatapath="[xpath]" and ftype="[fieldtype]"

Possible attributes

ftype="[fieldtype]" (line|text|enum|date|int)

dttype="[datatype]" (string|int|date|datetime|time|html)

dtminlength="[minlength]"

dtmaxlength="[maxlength]"

dtrequired="[true|false]"

rows="[rowcount]"

Needed childnodes

None

Possible childnodes

<prompt /> For every field, a prompt-text can be given. If defined, the prompt text will be visible in front of the field in the wizard.

<description /> For every field, a description can be given. If defined, the description will be shown "onmouseover". If the user move the mouse over the field, the description will be shown in a hint.

Possible parents

<form-schema />

<item />

Example

```
<form-schema id="step1">
  <title>Example form</title>
  <field name="title" dtminlength="1" ftype="line">
    <prompt>News Title</prompt>
    <description>You can enter the news-title here</descrip
  </field>
</form-schema>
```

field ftype="line"

Syntax

<field ftype="line" />

Usage

A line-field will show a single-line inputfield in the wizard. Use them for simple text entry.

Needed attributes

None

Possible attributes

dttype="string"

dttype="int"

	dtminlength="[minlength]"
	dtmaxlength="[maxlength]"
	dtrequired="[true|false]"
Needed childnodes	None
Possible childnodes	see field (general)
Possible parents	see field (general)
Example	

```
<field name="title" dtminlength="1">
  <prompt>Title</prompt>
</field>
```

field ftype="int"

Syntax	<field ftype="int" />
Usage	Use this field for number editing
Needed attributes	None
Possible attributes	dtmin="[minvalue]"
	dtmax="[maxvalue]"
	dtrequired="[true|false]"
Needed childnodes	None
Possible childnodes	see field (general)
Possible parents	see field (general)
Example	

```
<field name="score" dtmin="100" dtmax="5000" dtrequired="tr
  <prompt>Enter position</prompt>
  <description>Enter value between 100 and 5000 please.</de
</field>
```

field ftype="date"

Syntax	<field ftype="date" />
Usage	Use this field to edit date, date-time, or time fields.
Needed attributes	None
Possible attributes	dtmin="[mindate]"

	dtmax="[mindate]"
	dttype="[date|datetime|time]"
Needed childnodes	None
Possible childnodes	see field (general)
Possible parents	see field (general)
Example	

```
<field name="start" dttype="date">
  <prompt>Startdate</prompt>
</field>
```

field ftype="upload"

Syntax	<field ftype="upload" />
Usage	Use this field to process uploads. Note: Make sure that you use this field in the right context: Usually, this field will store it's binary-value in a mmbase field named 'handle'. See the upload example for more info.
Needed attributes	None
Possible attributes	None
Needed childnodes	see field (general)
Possible childnodes	see field (general)
Possible parents	see field (general)
Example	

```
<wizard-schema>
  <title>Image Upload</title>
  <action type="create">
    <object type="images" />
  </action>
  <action type="load">
    <field name="title" />
    <field name="description" />
  </action>
  <form-schema id="step1">
    <title>Image upload</title>
    <field name="title">
      <prompt>Title</prompt>
    </field>
    <field name="description" ftype="text" rows="8">
      <prompt>Description</prompt>
    </field>
    <field name="handle" ftype="image" dttype="binary">
      <prompt>upload</prompt>
    </field>
  </form-schema>
</wizard-schema>
```

field ftype="startwizard"

Syntax	<field ftype="startwizard">
Usage	Use this field to start one wizard, from inside another wizard.
Needed attributes	objectnumber="{object/@number}"
	wizardname="[name of the wizard]"
Possible attributes	inline="[true|false]" an inline startwizard will replace the current wizard to create the new object and come back when ready, a not-inline startwizard will pop-up a window to create the new object .
Needed childnodes	None.
Possible childnodes	None.
Possible parents	<item />
Example	

```
<command name="startwizard" inline="false" wizardname="task
```

field ftype="data"

Syntax	<field ftype="data" />
Usage	Use this field to show values, but don't make it editable. In other words: use this to make a read-only field.
Needed attributes	None
Possible attributes	see field (general)
Needed childnodes	None
Possible childnodes	see field (general)
Possible parents	None
Example	

```
<field name="firstname" ftype="data">
 <prompt>Your firstname is:</prompt>
</field>
```

field ftype="enum"

Syntax	<field ftype="enum" />
Usage	Use this field to make a "dropdown" inputfield. (In HTML-terms: a selectbox).
Needed attributes	None

Possible attributes	None
Needed childnodes	<optionlist /> Use this node to define the possible options for the field. (see also: optionlist)
Possible childnodes	see field (general)
Possible parents	see field (general)
Example	

```
<field name="type" ftype="enum">
  <prompt>Articletype</prompt>
  <optionlist name="articletypes">
    <option id="1">News</option>
    <option id="2">Interview</option>
    <option id="3">Information</option>
  </optionlist>
</field>
```

lists

Syntax	<lists />
Usage	Use this node to define optionlists in a wizard.
Needed attributes	None
Possible attributes	None.
Needed childnodes	<optionlist /> (see also:optionlist)
Possible childnodes	See needed childnodes
Possible parents	<wizard-schema />
Example	

```
<lists>
  <optionlist name="priorities">
    <option id="1">Low</option>
    <option id="2">High</option>
  </optionlist>
</lists>
```

optionlist

Syntax	<optionlist />
Usage	Use this node to define an optionlist. It can be either a static list of options, or a list filled using an MMBase query.
Needed attributes	None
Possible attributes	name="[optionlistname]" This optional attribute can be used if an optionlist is defined in a <lists /> node. To reference to the defined op-

tionlist, you will need the optionlistname. Eg.: <optionlist select="earlier_defined_optionlist" />

optionid="field[@name='fieldname']" This optional attribute can be used if an underlying 'query' node is present. Of all matching nodes, the field with name given in 'optionid' will be used as the 'id' part of the option.

optioncontent="field[@name='fieldname']" This optional attribute can be used if an underlying 'query' node is present. Of all matching nodes, the field with name given in 'optioncontent' will be used as the content part of the option.

Needed childnodes

<option id="[optionid]">[optionvalue_to_be_shown_in_wizard]</option>

Possible childnodes

<query> This childnode can be used instead of <option> childnodes to fill the optionlist with options. The query has three attributes: optional 'where' and 'orderby' clauses and a required 'xpath' clause. The xpath can have the following syntax:

- /*@buildername, in this case a nodequery will be performed.
- /*@buildername,otherbuildername,..., in this case a multilevel query will be performed. When doing a multilevel query, there must be a <object> childnode with two <field> childnodes for the 'optionid' and 'optioncontent' arguments of the optionlist.

Possible parents

<wizard-schema />

<field ftype="enum" />

Example

```
<optionlist name="role"
            optionid="field[@name='role.role']"
            optioncontent="field[@name='role.name']">
    <query xpath="/*@companies,roles" where="company.number
        <object>
            <field name="role.role" />
            <field name="role.name" />
        </object>
    </query>
</optionlist>
```

command (name="search")

Syntax

<command name="search" />

Usage

Use this command inside a list node to define how a user can search for objects and add them to a list.

Attributes (general)

See for detailed information how to use nodepath, startnodes, fields, constraints, orderby the documentation of the MMBase taglib.

Needed attributes

name="search"

nodepath="[buildername_to_start_with]"

	fields="[fieldnames of fields to show]"
Possible attributes	startnodes="[objectnumber]"
	age="[default age to show in search-field]" use -1 to set it to "any age", use 1,7,31,365 for day, week, month, year ages.
	constraints="[mmbase where part]"
Needed childnodes	None
Possible childnodes	<prompt /> Place the text to be shown in front of the searchfield here.
	<search-filter /> Defines what extra searchfields should be shown and used in the query. See also: search-filter.
Possible parents	<list />
Example	

```
<list destination="people" minoccurs="0" maxoccurs="*">
  <command name="search" nodepath="people" fields="firstna
    <prompt>#Search command prompt</prompt>
    <search-filter>
      <name>firstname contains</name>
      <search-fields>firstname</search-fields>
    </search-filter>
    <search-filter>
      <name>lastname contains</name>
      <search-fields>lastname</search-fields>
    </search-filter>
  </command>
  <item>
    <field name="firstname" >
      <prompt>Field prompt</prompt>
      <description>Field Description</description>
    </field>
  </item>
  <action type="create">
    <relation destinationtype="people" role="related" />
  </action>
</list>
```

search-filter

Syntax	<search-filter />
Usage	Use the searchfilter to allow the user to fire a free-text query from the wizard. See the command name="search" example.
Needed attributes	None
Possible attributes	None
Needed childnodes	<name />
	<search-fields />
Possible childnodes	None

Possible parents	<command name="search" />

Example

```
<search-filter>
  <name>Naam bevat</name>
  <search-fields>name</search-fields>
</search-filter>
```