XIOS/3 Development Manual

Generated 06/09/2022

Table of Contents

Table of Contents 2
XIOS/3 Development 8
Launching XIOS/3 8
Application Development 9
The MVC Design 9
Structure 9
Data XML (Model) 10
UI XML (View) 10
Process XML (Control) 10
Introducing the XIOS/3 Filesystem 11
File Properties 11
System Permissions 11
Performing Actions 11
Using the XIOS/3 Filesystem 12
Overall Structure 12
Managing Files 12
Managing Folders 15
Managing Groups 17
Managing File Types 20
Searching the Filesystem 20
Development Tools 21
XIOS Applications 21
Icons and Images 21
Application Packaging 21
Overall Structure 22
Regular Package 23
Inline Package 24
Widget 25
Extending XIOS/3 26
Additional Components 26
Additional Operations 28
UI Development 28
Overall Structure 28
View 28
Toolbars 30
Menubars 30
Layout 30
Using Components 30
Types 31
Attributes 31
Attribute values 31
Common attributes 32
Events 33
AlignCenterRelease 33
AlignCenterSelect 33
AlignJustifiedRelease 33
AlignJustifiedSelect 33
AlignLeftRelease 33
AlignLeftSelect 33
AlignRightRelease 34
AlignRightSelect 34
Attached 34
BGColorRelease 34
BGColorSelect 34
Blur 34
BoldRelease 34
BoldSelect 34
BulletListRelease 34
BulletListSelect 34
Change 34
Changed 34
Click 35
Close 35
Closing 35
Complete 35
Completed 35
ContextMenu 35
DateChanged 35
DecorationRelease 35
DecorationSelect 35
Delete 35
Detached 35
DoubleClick 35
drawingComplete 36
DrawMode 36
drawModeOff 36

Drop 36
Edit 36
EditEnd 36
End 36
Empty 36
Escape 36
FGColorRelease 36
FGColorSelect 36
Focus 36
FontFamilyRelease 37
FontFamilySelect 37
FontSizeRelease 37
FontSizeSelect 37
Fullscreen 37
Header 37
HeaderContextMenu 37
Home 37
Hover 37
HttpHyperlink 37
Init 37
ItalicRelease 37
ItalicSelect 38
Key 38
Loading 38
Ready 38
Menu 38
MediaEnded 38
MediaRange 38
Move 38
MoveItem 38
Next 38
NumberListSelect 39
Paste 39
Previous 39
Release 39
Return 39
Saved 39
Select 39
SelectedDate 39
SelectMode 39
SelectedMore 39
SetActive 39
Show 39
SubjectMore 40
Tab 40
TriggerEvent 40
UnSelect 40
UnSelection 40
UpdateTime 40

Actions 40

activate 40
add 40
clear 41
disable 41
enable 41
hide 41
open 42
readonly 42
readwrite 42
release 42
select 42
setValue 43
show 43
unbind 43

Methods 43

activate 43
adjustVolume 44
appointmentColor 44
attach 44
callFlash 44
changeInterval 44
clear 45
clearContent 45
clearSearchLines 45
copyToClipboard 45
deleteShape 45
deselect 45
detach 46
disableAutoSpellcheck 46
edit 46
enableAutoSpellcheck 46
execCommand 46
executeCommand 46
executeScript 47
flashTab 47
focus 47

focusItem 47
focusSelection 47
getCurrentLabel 47
getCurrentPosition 48
getDataLocation 48
getDateValue 48
getDimension 48
getEndDate 48
getEndTime 48
getLat 48
getLineBottomMargin 49
getLineTopMargin 49
getLng 49
getMapType 49
getZoom 49
getNodePath 49
getParam 50
getPaste 50
getSelected 50
getSelectedItem 50
getSelectedText 50
getSelection 50
getSelectionLength 50
getSelectionValue 51
getSrcDataLocation 51
getStartDate 51
getStartTime 51
getText 51
getUpdateEnabled 51
getValue 51
getVolume 51
gotoLine 52
hideColumn 52
haveSelection 52
insertHyperlink 52
insertImage 52
insertText 52
lineNumbers 53
maximize 53
minimize 53
modStyle 53
navigateUp 53
next 53
nodeExpanded 54
pasteFromClipboard 54
pause 54
play 54
previous 54
print 54
printPresentation 55
release 55
releaseEnter 55
removeList 55
repeat 55
resetSelection 55
restore 55
restoreAppDlg 56
scale 56
search 56
searchMap 56
selectAll 56
selectFirstRow 56
selectLastRow 57
selectTab 57
setBulletType 57
setDimension 57
setEditableMode 58
setFont 58
setIcon 58
setImage 58
setLabel 58
setLineMargin 58
setLineIndent 59
setLook 59
setMode 59
setMute 59
setParam 59
setRange 59
setSelect 60
setSelectedDate 60
setSelectionEx 60
setSelectionRoot 60
setSkin 60
setTitle 60
setType 61
setUpdateEnabled 61
setValue 61

setView 61
setVolume 62
setWeekStartDay 62
showColumn 62
showControls 62
shuffle 62
sortColumn 62
stop 63
takeEnter 63
toggleCheckable 63
toggleNode 63
update 63
updateData 63
updateRedraw 64
wordWrap 64
Rules 64
Look and Feel 64
Process Development 65
Overall Structure 65
Process 65
Triggers 66
Steps 66
Aliases 67
Alias Types 68
System Aliases 69
Operation 70
Event Model 70
Component Events 71
View Events 72
System Events 73
Using Expressions 73
Definition 73
Using Operations 74
Attributes 75
Examples 75
Applications 75
Book List 75
Layout 77
Column Layouts 77
Mixed Layouts 78
Row Layouts 79
References 80
ActionUrls 80
lastLogins 80
leaveGroup 80
search 80
searchUser 81
Components 81
Accordion 81
Breadcrumbs 82
Browser 83
Button 84
ButtonBox 84
Calendar 86
CheckBox 87
Clock 88
Color 89
ComboBox 90
Container 92
DataItem 92
Date 93
Divider 94
Form 95
Grid 96
Group 98
Image 99
Input 99
Item 101
List Item 101
Menu Item 101
Plate Item 102
Label 103
List 104
ListView 106
Map 108
Media 109
Menu 110
MenuBar 111
Panel 112
Plate 114
Radio 115

Render 116
RichText 117
Shell 118
Slider 120
TabStrip 121
TextArea 122
ToolBar 123
Tree 124
Functions 125
apply-locale 126
capitalize 126
concat 126
count 126
count-selection 126
changes 127
channelname 127
contains 127
document 127
downcase 128
escape-string 128
filename 128
filetype 128
folderexist 129
folderid 129
foldername 129
folderpath 129
format-bytes 130
localizedate 130
formatdate 130
get-label 131
index-of 132
is-alias 132
is-open 132
is-running 132
local-name 133
match 133
parsedate 133
position 133
random 134
read 134
replace 134
selected-text 134
selection 135
substring 135
trim 135
trim 136
upcase 136
xml 136
Process Operations 136
action 136
bind 137
call 138
callMethod 139
callService 140
change 141
close 143
create 143
channel 144
debug 146
decision 146
filesystem 147
foreach 152
friends 153
groups 153
login 154
navigate 155
open 156
print 157
rollBack 158
setSelection 159
save 159
share 160
split 161
Dictionary 162
atom namespace 162
CSS Syntax 162
Content-Type 162
data transaction 163
dc namespace 163
fs namespace 163
l namespace 163
local-name 163
mutex 163
MIME type 163

ni namespace 163
text() 163
tool-tip 163
xlink namespace 163

XIOS/3 Development

Development and reference manual for XIOS/3. For an easier to digest introduction to XIOS/3 please have a look at the XIOS/3 Tutorial.

What is XIOS/3?

XIOS/3 is the world's most advanced environment for developing and running web applications. It allows you to use XML as the template language to build visually rich applications in a higher abstraction level, with great capabilities of interacting with any server back-end resource or code. One way of describing XIOS/3 is as a Edge Application Server, an Application Server on the client side used as a user interface front-end for a Service Oriented Architecture (SOA).

Having a Edge Application Server at your disposal means that most of your application logic can be performed directly on the client, with a minimal amount of server requests as an end result. The applications are executing within the web browser of the client, making XIOS/3 an ideal partner with any server technology. Utilizing resources on the client, with reduced server dependency, drastically reduces the perceived latency in web applications with a better user experience as a result.

XIOS/3 is designed to be what HTML would have been, if HTML had been designed for developing and running applications, not text-based content. Using HTML with a JavaScript framework on top only gets you a bit on the way of creating rich applications, but if you want to develop applications with usability close to installed software, XIOS/3 is the most efficient way.

A Complete Client Side Solution

The XIOS/3 technology is entirely located on the client side. It handles everything from the user interface and logic to communication and underlying backend systems. It's an application environment that provides a complete solution from UI components, through event routing, data flow management and transaction engine, to service abstractions and protocol management. XIOS/3 is self contained with no server components and also makes it possible to integrate to existing systems without requiring any changes to them.

What the Documentation Covers

This documentation is an introduction to XIOS/3 concepts, application development and packaging. In addition it contins a reference section with all the included events, components and expression functions. Finally a collection of small examples. The first section you are reading now contains basic information on how to quickly set up XIOS/3 for development and production deployment. The following chapter is the Application Development section where you are introduced to how basic application development works and example of applications. This is followed by Application Packaging describing how to create application packages for deployment. Next is the UI Development section, describing the view part of MVC and how components are combined together. This is followed by the Process Development chapter describing the Controller part. Finally is the reference sections of the manual.

Quick Start

To start using XIOS/3, take the zip archive and unzip it in the root of a webserver. Then go with your web browser to the location of your web server and load the index.html file. Once loaded you are now running the XIOS/3 shell application. From here you can load any of your own projects, e.g. "load demo.xml". Type "help" to see a list of options available in the shell application.

During development of XIOS/3 applications it may be convenient to have a local webserver running on your development computer. This can host both XIOS/3 and the application you are writing. All of XIOS/3 code is contained within the "xios" directory, making it easy to drop XIOS/3 into any existing web server without conflict. It is however possible to host the web page on one server, XIOS/3 on another server and the application on a third.

There are many webservers available to chose from, but one easy option available everywhere Python is installed, such as MacOS and most Linux systems, is to use its built in web server. You can start it by opening a command shell in the directory you want as the root directory for your web server, and type the following to open a web server on http://localhost:8080/

```
python -m SimpleHTTPServer 8080
```

Launching XIOS/3

XIOS/3 is loaded and initialized in multiple steps to support a distributed and flexible architecture. The first step is an HTML file that sets up the initial loading conditions for the environment, both in term of fundamentals such as what the home domain name should be and if HTTP or HTTPS is used, but also to give the developer the final say in how XIOS/3 is loaded.

In the HTML page the boot parameters for XIOS/3 is defined in the JavaScript object "boot". Then, when all deployment specific additions are made, XIOS/3 bootloader is loaded into the page. It will load and start XIOS/3 according to the boot object.

HTML Loading Page

Minimally the HTML loading page only needs to load the boot loader, and it will default to load the shell application.

```
<html>
  <head>
    <meta charset="utf-8" />
  </head>
  <body style="overflow: hidden;">
    <script src="xios/boot.js" charset="utf-8"></script>
  </body>
</html>
```

Example: Minimal HTML loading page

Every deployment of actual code projects requires at least that XIOS/3 is given an application to start executing, instead of loading the shell application. This is controlled by the startup member of the boot object. To load the example startup process "xios/config/startup.xml", att the following script element to the HTML header.

```
<script>
var boot = {
  startup : "xios/config/startup.xml"
};
</script>
```

Example: Define boot and load startup

The "xios/config/startup.xml" file accomplish two things. It defines a new protocol in XIOS/3 under the name "https" using the HTTPS protocol module. This allows you to load data using https://-URLS within XIOS/3. Having done that it starts the shell application "xios/apps/shell/shell.xml".

What should go into the startup document depends on your project. If you are making a stand alone application, you can point XIOS/3 startup sequence to the main document directly. If you want to separate between the XIOS/3 environment and the applications running within it, it could make sense to limit the start script to initialization and then hand over to external applications or another loader.

```
<process>
  <step id="1" name="Start Shell">
    <operation name="channel">
      <name>https</name>
      <title>HTTPS</title>
      <protocol>HTTPS</protocol>
    </operation>
    <operation name="open" value="xios/apps/shell/shell.xml"/>
  </step>
</process>
```

Example: startup.xml

Whenever network activity occurs XIOS/3 will make the DOM element "activityLayer" visible. This allows you to design overlays that signals load activity.

```
<div id="activityLayer"
  style="position: absolute; right: 50px; bottom: 50px; z-index:9999;">
  <img src="loading.png" style="opacity: 0.7;">
</div>
```

Example: Network activity overlay

Application Development

In this section we will cover everything about XIOS/3 Application Development and it is divided in to sections of relevance. The very foundation of Application Development in XIOS/3 is the fact that they are XML based and that they use the MVC design pattern, which means that an application consists of three parts; the UI, the Logic and the Data and that these are kept separate. This design pattern makes development more efficient while simplifying parallel development where developers may work separately on the different parts of the application.

What this section covers

This section covers the foundations of XIOS/3 Application Development and introduce the MVC design model.

This section also covers an explanation to the Application Package, which is explained in detail with examples. The concept of Application Package allows the files to be bundled in to an application package, and then treated as that by the XIOS/3 system.

The final chapter is devoted to application examples. A section that will be updated continuously with more examples by XIOS/3.

The MVC Design

XIOS/3 is designed with the MVC (Model/View/Control) design model in mind. The user interface, the data and the logic of a component are defined in separate files, all using XML.

The idea behind the MVC design is to separate data (the model) from the user interface (the view). An intermediate layer (the control), the code or programming is responsible for the logic and links together the data and the user interface. This design simplifies updating of the system. It also enables the use of, for example, one component's logic in another component or having multiple views of the same application.

The model

The model is described in standard XML and is the data that gives the graphical component information. This information is presented to the end-user as text, numbers, pictures or other informative data.

The view

The view is described in XIOS/3 UI XML and presents the interface, creating an application view containing graphical components. Each component is described in XML as an element, similar to the way a HTML tag describes a graphical widget for a website.

The control

The control is described in XIOS/3 process XML and is an abstract way of programming. It is the application logic - an intermediate layer connecting the model and the view.

The process XML consists of a number of steps, and typically execution starts on step 1. A step consists of operations changing application states, application user interactions or data storage to the server. Most steps are invoked by something called a trigger.

A trigger is an event caused by user interface components invoked when the end-user interacts with the application, for example using the keyboard or mouse. The XIOS/3 system sometimes invokes system events that can be connected to a trigger as event types.

Structure

This section explains what a XIOS/3 application consists of and briefly explains XML syntax. The latter is very important since XIOS/3 applications are built using XML and we have therefore compiled a short list of restrictions and a basic syntax of XML.

In order to develop XIOS/3 applications you also need to understand the application MVC design model, be familiar with the application package structure. How an application structure works will be explained in the section Creating and packaging applications.

An application consists of at least one application view, but can have any amount of views. A view is a window or popup part of an application.

This section will conclude with a simple example application consisting of one view managing a list of books.

The three elements of the MVC design are always divided in to different files, separating the model, view and control from each other. Next, we will explain how to define and work with the model to build XIOS/3 application views and make the application come alive.

XML Syntax and Restrictions

The XML syntax is very easy and logical. These XML restrictions apply to all XML languages, such as XML, XSLT, XHTML, UI XML and Process XML. The following rules and restrictions apply to all XML languages.

- Every XML document must have a XML processing instruction.
- Every XML document must have a root element.
- An element name must never start with the word XML.
- XML is case sensitive, there is a difference between <a> and <A>.
- Only empty elements should be self-closing.
- All attributes should be places inside double or single attributes. They can however not be used at the same time.
- Elements may not overlap each other.

The last rule which states that elements must not overlap each other means that; if B is the child element of A then B must be closed before A and never after else wise they would overlap each other.

```xml
<?xml version="1.0"?>
<element attribute="A" attribute2="100">
  <childElement value="1"/>
  <childElement value="2"/>
  <childElement>No Value</childElement>
</element>
```

Example: Basic XML Syntax

Data XML (Model)

The data XML has no restrictions on element structure except that it must be well-formed XML. The data XML can be structured in several files; these can be static - or dynamically generated files requested from the server via the client web browser.

```xml
<?xml version="1.0"?>
<documentRootName>
  <!-- Put XML document elements here -->
</documentRootName>
```

Example: Data XML basic structure

UI XML (View)

The UI XML describes an application view structure containing layout elements and graphical components. All of these are elements in the UI XML, with names corresponding to the names of the respective application view or components.

This section will only cover the UI XML language in brief. A more detailed explanation of element restrictions, required and optional attributes and other related application configuration can be found in the section on UI Development and in the Component Reference section.

```xml
<?xml version="1.0"?>
<view name="[name]" version="[version]" icon="[path]">
  <panel name="[name]" width="auto">
    <!-- Put application view panels and components here -->
  </panel>
</view>
```

Example: Basic XIOS/3 application view structure in a UI XML file

View element

The application view UI XML must have a View root element. The element must specify the name and version attributes. The name attribute must be a unique name identifier and the version attribute must specify the version of the UI components to be used.

Panel elements

To describe the application view layout, a single top-level panel element must be placed inside the view element. The container panel can contain any amount or structure of panel layout elements or component elements.

Component elements

The component elements are the actual visual building blocks placed inside the view layout panels. They visualize the common components of almost any modern application, GUI library or internet related system. For example list, tree, menu and toolbar components are widely used in XIOS/3 applications.

Process XML (Control)

The Process XML describes the application logic execution steps and the panel element operations performing the actual programming actions for each step. The operations let you control the application; you could compare operations to functions in traditional programming languages.

This section will only cover the Process XML language in brief. A more detailed explanation of element restrictions, required and optional attributes and any other related application logic programming can be found in the section on Process Development and in the Process Operations Reference Manual.

```xml
<?xml version="1.0"?>
<process name="[name]" version="[version]">
  <!-- Put your triggers here -->
  <trigger view="[name]" component="[name]" event="[type]" step="[unique identifier]"/>

  <!-- Put your steps here -->
  <step id="[unique identifier]" name="[name]">
    <!-- Put your operation code here -->
  </step>
</process>
```

Example: Basic XIOS/3 process structure in a process XML file

Process root element

The process XML must have a process root element. The element must specify the name and version attributes. The name attribute must be a unique name identifier and the version attribute specify the version of the process language to use.

Trigger element

At the top of the process XML you will find trigger elements. A trigger is an event caused by user interface components invoked when the end-user interacts with the application, for example using the keyboard or mouse. A trigger can also be invoked by actions performed by the XIOS/3 system. Triggers initiate at a specific step and start executing operation.

The trigger elements have to specify:

- view attribute, referring to the application view to which the trigger is mapped
- component attribute, referring to the GUI component to which it is mapped
- event attribute: the component or system event type to get triggered
- step attribute: the step to handle the event triggered

Step element

Step elements follow the trigger elements. They provide instructions on what should occur when the trigger has been invoked. The step element may contain any amount of alias elements or operation elements. A step can continue executing operations in another step by specifying a goto attribute with the number of the next step as value. Movement to another step will take place when all operations in the first step have been executed.

The step elements have to specify:

- id attribute holding the identifier of the step
- name attribute specifying a unique name of a step

Alias element

An alias is a variable (global or local) in the process XML file and will exist as long as the process is running. The alias can hold text, numbers or a document reference. Using aliases for document references makes reuse of references efficient, because there is no need to type the entire path every time a document must be accessed. If you want to use a reference to an alias, put a $ sign before the alias name.

Operation element

The operations are the actual programming logic and make the GUI come alive through end-user interactions. The process XML operations may remind you of method calls invoked on the components, symbolizing classes or objects in other programming languages.

Introducing the XIOS/3 Filesystem

The XIOS/3 filesystem is a filesystem abstraction for local and remote document storage and services. The filesystem abstraction provides generic methods to work with data stored in a wide range of different systems. Note that all operations are not available on all filesystem backends, but depends on the inherent constraints and functionality of the underlaying storage as well as the feature set of the protocol module that communicates with the storage.

Many of the system functions and applications in XIOS/3 assume that the default filesystem is named "home://". There is however no requirement that any filesystem is mounted in XIOS/3.

After reading this section you should have basic understanding how file types in XIOS/3 work and how to associate file types while also knowing that there are existing file types and where they are located.

We will also provide you with a overlook of file properties and system permissions. These are intended to educate the reader of available file properties set by the system such as when the file was created as well as settings or limiting permissions to files by making them read-only for other users. Again this is intended to give the reader an introduction to the filesystem and does not offer any code examples. These can instead be located in the "Using the XIOS/3 Filesystem" section.

This section also introduces the three different actions that can be performed on files, open, edit and preview. These can be found in the "Performing Actions" section.

File Properties

All files have their meta data exposed through the virtual ".meta" document in the filesystem. In the filesystem the actual file is treated as a directory in which the ".meta" file can be found. For example the file "public.xml" has its meta data file readable at "public.xml/.meta". The contents of the meta data file is conforming to the W3C Atom metadata syndication format.

The properties below are available as attributes in the meta data XML listing of a file.

Created

The creation time of a file is available in the atom:published element of the metadata document. The text content will be a timestamp in the format of YYYY-MM-DDThh:mm:ssZ, e.g. 2019-01-17T13:25:57Z.

Modified

The time of last modification is abailable in the atom:updated element of the metadata document. The text content will be a timestamp in the format of YYYY-MM-DDThh:mm:ssZ, e.g. 2019-01-17T13:25:57Z.

System Permissions

The XIOS/3 system offers file permission handling for local files as well as a user/group permissions for the Group functionality.

Group permissions

Each group can have three general permission levels (customized roles are covered in the Managing groups section). The general roles are Owner, Administrator and Member which all have predefined access rights in the group.

The owner is a super-user that have full rights across the system. Usually the owner is the creator of the group. Furthermore the owner is also the only user that can promote and demote users to become administrators. These have mroe access rights than normal users in the group. Administrators role is created automatically when the group is created.

Normal members have access to the filesystem but can't create or delete files without the owner or admin setting special permissions for files or folders. Observe that files inherit the rights of the folders, if they do not have their own rights defined.

Additional

Files created in XIOS/3 can handle basic read, write and delete permissions. This can be done by modifying the meta data of the file in question. These permissions can also be changed using Process XML.

Performing Actions

Actions can be performed on files which are handled by operation open. Information regarding which application should be used to perform a specific action on a file is set in the settings.xml file.

All file types can be either opened, edited or previewed. Due to the fact that XML files can be defined in many different ways these actions can be used to sort them out and open different applications for different XML files.

Open

The open actions handles which application should be used to open the file. It actually works by mime-type and root element of the file. In order to open an XML file with a specific application the file type must be defined in settings.xml else it will open in the default application for XML files, Notepad.

Edit

This option refers to which application should handle the editing process of the file. Many times this would be the same application that is used to open the file. There are however exceptions such as View XML and Process XML files or any files that are meant be edited by an editor while displayed in a completely different way.

Preview

This action is also optional and is used as a third option of opening a file.

Using the XIOS/3 Filesystem

This section of the documentation contains information about the XIOS/3 filesystem.

In the chapter Overall Structure we explain how the filesystem works and of what it consists of.

Managing the filesystem using Process XML to handle Files, Folders and Groups is covered extensively with several examples provided. The final chapter covers how to perform searches in the filesystem.

Overall Structure

The XIOS/3 filesystem utilizes XML for communication. Every folder content request is returned as an atom feed which is similar to an RSS-atom feed that websites use to syndicate news on websites. In the application Document Explorer this atom feed can be visualized in several different ways.

In this section we will more closely explain how the different components of the filesystem function.

Files

All files are stored on the server that the protocol module connects to. Upload is possible via either the Document Explorer or by using a an alias (model open).

In folder listings and search queries each file is presented as an atom:entry in an atom:feed. In this atom:entry element you are provided with the location of the file, MIME-type and other file specific information.

To learn more about how to use atom feeds read the Managing Files chapter. Information about how search queries work can be found the Searching the Filesystem chapter.

Folders

All folders are virtual and the entire folder structure is stored in the file folders.xml. It is this file that is used to render the folder structure in applications such as the Document Explorer.

In XIOS/3 folders use the fs namespace. This means that this namespace must be declared in any file that is used to modify file structure or display it.

Shortcuts

A user can create a shortcut to any file by right-clicking on a file and selecting the "Add shortcut to desktop"-option in the context menu. Any user can also right-click directly on the desktop to create a "shortcut" or an "iLauncher Shortcut". The latter is used for external application or mail services. The "shortcut" option requires a Path to the application file and can also be provided a parameter in the form of a trigger.

XIOS/3 also supports folder shortcuts; this is done by pointing the shortcut to a folder and ending the destination URL with a slash like such:

    home://Documents/

The final slash is very important as it indicates if it is a file or a folder.

The shortcut functionality works by adding a atom:entry to the settings.xml file. Observe that the atom namespace is used.

Groups

Groups are a special feature in XIOS/3 where, when supported by the uderlying filesystem, a group is represented with a separate file system where documents can be shared within that group.

More information about creating and managing groups can be found in the Managing Groups chapter of this section.

Managing Files

This chapter is intended to provide the reader with the knowledge how the XIOS/3 filesystem handles files and how to create, edit and delete files by altering the atom feed.

Name Convention

The following characters are forbidden to use in file names:

    ? / \ ( ) # * : [ ] | % & " ' ! { }

Creating files

Files are created by appending an atom:entry element to an atom:feed. The feed is retrieved when making a folder listing or query in a specific folder. To append the XML data we must use the change operation.

Creating a file

```
<operation name="filesystem" value="home://Documents/">
  <create type="file" name="MyNewDocument.xml">
    <author>
      <name>Alice Wonder</name>
      <email>alice@wonderland.com</email>
      <uri></uri>
    </author>
    <summary></summary>
    <acl>
      <user id="101" read="true" write="true" delete="false" changeACL="false"/>
    </acl>
    <index>
      <key name="keywords" index="true"></key> <!-- will be indexed as dc:keywords -->
      <key name="site" index="false"></key>    <!-- will be indexed as ni:site -->
    </index>
    <content type="text/xml">
      <newConent>My New File...</newConent>
    </content>
  </create>
</operation>
```

Example: The operation will create a new document called MyNewDocument.xml in the Documents folder.

The above Process XML code will use the filesystem operation to save a document named "MyNewDocument.xml" in the "home://Documents/" folder.

Using this operation you can easy create a file, set the content, set the permissions rights and set the meta data (atom, dc and ni). In the author element which corresponds to the atom:author element we populate the child element's name, email and uri which will then be available as atom:name, atom:email and atom:uri as meta data. The same is true for the summary element which will be available as atom:summary in the meta data.

The acl element contains the permission rights for the file. You can add either user or group elements and give them specific rights. You can define the read, write and delete rights but also the right to change the ACL information.

The index element defines the meta data keys (dc: and ni:). If you set the index attribute on the key element to true the keyword will be available as dc:keyword and will also be searchable when searching the filesystem. The ni: meta data is not searchable.

Finally the content element which allows you to directly define the contents of the file by adding it inside the element. The type attribute on the content element will define the content-type (mime-type) in the meta data.

The file created

The above code will generate a file called MyNewDocument.xml in the Documents folder (home://Documents). When opened the file should look like this.

Observe that when trying to rename a file to one name that is already being used by another file it will result in a system dialog notifying the end-user that there already is a filename with that name. Also observe that a file cannot be named exactly as a folder in the same directory.

```
<?xml version="1.0"?>
<newConent>My New File...</newConent>
```

Example: MyNewDocument.xml in home://Documents/

Handling file rights (file permissions)

File rights are appended to a file when it is created (see above) or by changing exiting file permissions. The acl element can have either user or group child elements. Both of these have id, read, write, delete and changeACL attributes which are used to define permissions.

The code below will give the a user with the id 20013 full permission for the file as well as changing the file rights (changeACL attribute). It also provides read access for all members of the group which has id 410. The members cannot write anything in the file or delete it.

```
<acl>
  <user id="20013" read="1" write="1" delete="1" changeACL="1"/>
  <group id="410" read="1" write="0" delete="0" changeACL="0"/>
</acl>
```

Meta Data

Meta Data is descriptive data about a file and its content. In XIOS/3 file meta data is appended as child elements to the atom:entry element. These child elements utilize three different namespaces; atom, dc and ni.

The atom-namespace is a W3C standard namespace which has predefined elements. These will be indexed and are searchable when making queries from the file server.

The other two namespaces are XIOS/3 namespaces. The dc-namespace (directory) is used to create meta data that should be indexed, and therefore searchable. The ni-namespace (no index) is used to create meta data that will not be indexed and will not be searchable when making server queries.

Developers can create and store their own meta data elements using the dc: and ni: namespaces. There are limitations to what the meta data field can be named. The local-name of the atom:, dc: and ni: elements must be unique and because the elements using the atom-namespace are reserved that means that dc:title or ni:title can't be used. The reason why the elements must have unique names is when making search queries the namespace is omitted.

A list of reserved element names using the atom-namespace can be found in the atom namespace dictionary entry.

The Shell application (or rather the shell component) has a command named "meta" that lists all meta data entries of a file. In the example below we will list all meta data of the Wallpapers.xml file.

```
meta home://Wallpapers.xml
```

Accessing meta data of a file

There are two different ways of accessing meta data of a file. The first is to create a file listing by searching for a file in a particular folder, thus generating an atom feed containing only one atom:entry element. In this case we will receive the meta data as atom, dc or ni namespaced elements.

The second way is to access the meta data for a specific file which is possible if the file path is known. In order to get the meta data we write the entire URL followed by a slash, like such:

home://Folder/folder/

The code below is the returned XML for Wallpapers.xml

```
<meta name="Wallpapers.xml" created="2019-01-28T11:42:10Z" href="http://os.xcerion.com/v1/5910/452165/1" src="home://Wallpapers.xml"
        size="89" mime="text/xml" schema="">
  <index>
    <key name="link" source="user">Wallpapers.xml</key>
    <key name="published" source="user">2019-01-28T11:42:10Z</key>
    <key name="updated" source="user">2019-01-28T11:42:10Z</key>
    <key name="id" source="user">mid:485@xios.xcerion.com</key>
    <key name="folder" source="user">5910</key>
    <key name="document" source="user">452165</key>
    <key name="content" source="user"></key>
  </index>
  <noIndex>
    <key name="root" source="user">wallpapers</key>
  </noIndex>
  <access tag="475d55G81-s23ssD">
    <user id="181" read="1" write="1" delete="1" changeACL="1"/>
  </access>
  <attachment>
    <document name="Wallpapers.xml" id="1" mime="text/xml" created="2019-01-28T11:42:10Z" modified="2019-01-28T11:42:10Z"/>
  </attachment>
</meta>
```

The code above is returned when using an URL similar to one in the debug operation below. The XML of the meta data is divided in to indexed meta and non-index meta data. These are presented as dc and ni namespaces in file listings. The access element contains information regarding which user can edit the file, in this case only one user.

```
<operation name="debug" value="home://Wallpapers.xml/">alert:plain</operation>
```

Changing meta data to a file

```
<operation name="change" value="home://Documents/">
  <store type="change" select="/atom:feed/atom:entry[atom:title='DocTestFile.xml']">
    <atom:title>Changed title</atom:title>
  </store>
</operation>
```

Example: Example of meta data

As mentioned earlier only meta data using the atom and dc namespaces are indexed and are searchable. For the example above you will be able to search for atom:title, atom:author and dc:fkeywords. To understand how searches are made read the "Searching the Filesystem" chapter.

The ni:developer will not be indexed and you will not be able to search for that information. You can however use it when it is available in atom:entry.

The name of the file created will be DocTestFile.xml and it will be created in home://Documents folder. The only contents of the file will be a single element (<document/>)

Attachments

XIOS/3 allows for XML documents to be attached to other documents. Currently only XML documents can be attached. This is accomplished by adding information to the meta XML of the file which the attachments will be a part of.

Because the meta data is available as XML data we use the change operation to append our attachment. In the value attribute we provide the meta data of the file taht we want to append the attachment to. This is done by adding one additional forward slash (/) to the file path.

All attachments of the file will be available in the attachments element as document elements. Note that there already exists one attachment with the id 1, which is the file itself. The id attribute is a positive integer, which means that the value must be a number higher than 0. Because the 1 is already in use the attachment can thus be 2 or higher.

When creating an attachment with an id that already exists it will overwrite any attachment that exist with the same id. It is important to understand that attachment cannot have meta data appended to them since they are only part of a document.

```
<operation name="change" value="home://file.xml/">
  <store type="append" select="/meta/attachment">
    <document name="My attachment" id="2"/>
  </store>
</operation>
```

Example: This will create an empty attachment to the document 'file.xml' and will have the name 'My attachment' and the id 2.

The attached file can be accessed by adding a forward slash and then the id within brackets like such:

home://file.xml/[2]

As you can guess this means that the original file can be accessed in one of two ways:

1. home://file.xml
2. home://file.xml/[1]

File listings

File listings are created every time you ask for the contents of a virtual folder and returned from the server in the form of an XML atom feed. File listing are also created when making a search query.

A folder listing looks as a normal folder path with an slash at the end of the Path. If the slash is omitted the Path will just be a string. When a slash is used it will return XML.

URL	Description	Result
home://Documents	Path to folder	string value
home://Documents/	file listing of folder contents	XML data

The atom:feed below is an example of a feed returned when asking for the contents of the folder home://. The root element is the atom:feed and notice the atom:, os:, dc: and ni: namespace declarations.

The os-namespaced elements at the top of the feed contain the number of search results. It is also for paginating when this is supported.

All files are listed as atom:entry node-sets. The feed below has lists three atom:entry node-sets and therefore three files. Each entry contains details about the file, such as filename (atom:title), when it was published and in what folder it is located.

```
<?xml version="1.0"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:os="http://a9.com/-/spec/opensearch/1.1/"
           xmlns:dc="http://xcerion.com/directory.xsd" xmlns:ni="http://xcerion.com/noindex.xsd">
  <os:totalResults>3</os:totalResults>
  <os:startIndex>0</os:startIndex>
  <os:itemsPerPage>100</os:itemsPerPage>

  <atom:entry>
    <atom:title>Wallpapers.xml</atom:title>
    <atom:published>2019-01-28T11:42:10Z</atom:published>
    <atom:updated>2019-01-28T11:42:10Z</atom:updated>
    <atom:link rel="alternate" type="text/xml" href="http://os.xcerion.com/v1/documents/5910/20086/1" length="89"/>
    <atom:id>mid:719e@xios.xcerion.com</atom:id>
    <dc:folder>5910</dc:folder>
    <dc:document>20086</dc:document>
    <dc:root>wallpapers</dc:root>
    <atom:content type="text/xml" src="home://Wallpapers.xml"/>
  </atom:entry>

  <atom:entry>
    <atom:title>folders.xml</atom:title>
    <atom:published>2007-08-30T06:51:10Z</atom:published>
    <atom:updated>2007-08-30T06:51:10Z</atom:updated>
    <atom:link rel="alternate" type="application/xml" href="http://os.xcerion.com/v1/documents/5910/12480/1" length="0"/>
    <atom:id>mid:5c8@xios.xcerion.com</atom:id>
    <dc:folder>5910</dc:folder>
    <dc:document>12480</dc:document>
    <atom:content type="application/xml" src="home://folders.xml"/>
  </atom:entry>

</atom:feed>
```

Example: File listing in home://

Renaming files

To rename a file using the process XML language all you need to do is to use to change operation to replace the old atom:title element with a new containing the new filename.

Observe that two files can't have the same name in a folder. If you try to rename the file to a file which already exists a system dialog notifying the end-user that the filename is in use will appear.

```
<operation name="change" value="home://Documents/?title:DocTestFile.xml">
  <store type="replace" select="/atom:feed/atom:entry/atom:title">
    <atom:title>AnotherName.xml</atom:title>
  </store>
</operation>
```

Example: Creating a file in home://Documents/

The example above makes a search query to the server for a file named DocTestFile.xml that should be located inside home://Documents/. It will return a atom:feed XML document. To better understand search queries, read the "Searching the Filesystem" chapter.

The feed returned will only contain one atom:entry since there can only be one file that is named DocTextFile.xml. We proceed to select the atom:title in the only atom:entry and replacing it with our new atom:title containing the text AnotherName.xml which will be the new name of our file.

Deleting files

To delete a file we need to remove it from the atom:feed. The example below will use the change operation and the child element delete to select a file named DocTestFile.xml and delete it.

Deleting file that do not exist does not create a system dialog notifying the end-user of the failure to remove it. It will only appear as a error message in the debugger, saying that the XPath way faulty and that it did not match anything.

```
<operation name="filesystem" value="home://Documents/DocTestFile.xml">
   <delete permanent="true" silent="false"/>
</operation>
```

Example: Creating a file in home://Documents/

Moving files

Moving files is a simple matter of using the filesystem operation.

Note: The path to the targetFolder must end with a slash.

```
<operation name="filesystem" value="home://Documents/DocTestFile.xml">
  <move to="home://Applications/" silent="true"/>
</operation>
```

Example: Moving a file to a new directory

Copying files

Observe that this will both copy and paste the file in the same folder. To work as default copy-paste it need to be broken up to two operations; "copy" to copy the filepath (targetFile) but you will also need to retrieve a target folder using alias model="folder".

Note: The path to the targetFolder must end with a slash.

```
<operation name="filesystem" value="home://Documents/DocTestFile.xml">
   <copy to="home://Copies/"/>
</operation>
```

Example: This will actually copy the file and rename it so that there is no name conflict

Uploading files

Uploading of files is done by using the filesystem operation.

```
<operation name="filesystem">
  <upload to="home://Documents/"/>
</operation>
```

Example: This will open a dialog window that will enable the user to upload a file in the Documents folder

Managing Folders

This chapter is intended to provide the reader with the knowledge to manage XIOS/3 folders. Working with folders means to work with folders.xml, a file created by the server which lists all your folders. The file not an actual file and can therefore not be removed.

To open you own folder.xml open the Shell application and type in the following:

```
xmlpad home://folders.xml
```

Note that all groups also have a folders.xml which is located in the folder root. This file is accessible to all group members.

home://folders.xml

In the folders.xml there is only one element, the fs:folder. Each folder-element can contain one or several folder-elements. Together they form a folder structure.

Each folder element has a id attribute, which is set by the server when the file is created and one name attribute which is given to the element when the folder is created.

This file utilizes the fs namespace and therefore such a declaration must be in any file that handles folder data.

Name	Type	Values	Description
id	numerical	[0...n]	server ID of folder

name	string	Any string value	Name of the folder
listview	string	URL	Specific XSL renderer for folder
path	string	path	Specifies the absolute path to the folder

```
<?xml version="1.0"?>
<fs:folder id="590" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:fs="http://xcerion.com/folders.xsd">
  <fs:folder id="592" name="Documents">
    <fs:folder id="2250" name="Pictures"></fs:folder>
  </fs:folder>
  <fs:folder id="602" name="Applications">
    <fs:folder id="4097" name="Booklist">
      <fs:folder id="4098" name="views"></fs:folder>
      <fs:folder id="4099" name="processes"></fs:folder>
      <fs:folder id="4100" name="data"></fs:folder>
      <fs:folder id="4101" name="layout"></fs:folder>
      <fs:folder id="4102" name="images"></fs:folder>
    </fs:folder>
    <fs:folder id="598" name="Trashcan" system="trashcan"></fs:folder>
    <fs:folder id="599" name="Groups" listview="apps/system/listviews/groups.xsl"></fs:folder>
  </fs:folder>
</fs:folder>
```

Creating folders

Folder are created by appending fs:folder-elements to the folders.xml document.

In our example below we define two aliases. The first, folderDoc, will contain the XML data stored in the folders.xml file. The second alias, folderName, contains the name of the folder which we wish to remove.

Name Convention

The following characters are forbidden to use in folder names:

? / \ ( ) # * : [ ] | % & " ' ! { }

We use the change operation change operation on the data in the alias folderDoc to append one fs:folder to the root element (/fs:folder[1]). The name of the new folder will be "My New Folder" just as we defined it in the alias folderName.

```
<alias name="folderDoc" value="home://folders.xml" model="data"/>
<alias name="folderName" value="My New Folder" model="value"/>

<operation name="change" value="$folderDoc">
  <store type="append" select="/fs:folder[1]">
    <fs:folder name="{$folderName}"/>
  </store>
</operation>
```

Handling folder rights

Folder rights can also be appended to a certain folder when it is created. By adding an access node-set as a child element of the fs:folder that we are trying to create we can define permissions. It is also possible to set permission rights on files. The access element can have either user or group child elements. Both of these have id, read, write, delete and changeACL attributes which are used to define permissions.

The code below will give the a user with the id 20013 full permission for the file as well as changing the file rights (changeACL attribute). It also provides read access for all members of the group which has id 410. The members cannot create or delete files in the folder just read the files. The changeACL attribute for the group element is set to 0 which stops the group users from changing the permission rights.

```
<operation name="change" value="home://folders.xml">
  <store type="append" select="/fs:folder">
    <fs:folder name="data">
      <acl>
        <group id="{$adminGroup}" read="true" write="true" delete="true" changeACL="true"/>
        <group id="{$userGroup}" read="true" write="false" delete="false" changeACL="false"/>
      </acl>
    </fs:folder>
  </store>
</operation>
```

Editing folders

There are restrictions to how folders can be edited. The id attribute which is used to identify the folder and the system attribute (which only exist on certain system folders) can never be changed. Thus the name attribute is used to rename folders.

The example below illustrates how to rename a folder named "New Folder" to "myFolder" by replacing the name attribute.

```
<alias name="oldName" value="New Folder" model="value"/>
<alias name="newName" value="myFolder" model="value"/>

<operation name="change" value="$folderDoc">
  <store type="replaceText" select="/fs:folder[1]/fs:folder[@name='{$oldName}']/@name">{$newName}</store>
</operation>
```

Deleting folders

Folder are deleted by removing their fs:folder-element from the folder.xml data.

In our example below we define two aliases. The first, folderDoc, will contain the XML data stored in the folders.xml file. The second alias, folderName, contains the name of the folder which we wish to remove.

We proceed to use the change operation to delete the fs:folder-element in the root that matches the XPath expression provided in the select attribute. This means that it will try to delete a folder named "My New Folder" that is located inside the root folder.

```
<alias name="folderDoc" value="home://folders.xml" model="data"/>
<alias name="folderName" value="My New Folder" model="value"/>

<operation name="change" value="$folderDoc">
  <delete select="/fs:folder[1]/fs:folder[@name='{$folderName}']"/>
</operation>
```

Moving folders

Moving folders is currently not supported.

Copying folders

Copying folders is currently not supported

Searching folders

The Process XML language allows you to search folders by making queries to the server. You can only search in one folder at the time. To learn how to make searches in folders read the "Searching the Filesystem" chapter.

Managing Groups

This chapter is intended to provide the reader with the knowledge to manage and understand the Groups functionality offered in XIOS/3.

Groups are a special feature in XIOS/3 that allows several individuals to share a common virtual disk. There exists two types of groups, public and private. Anyone can access the virtual disk of a public group, but to be able to gain access to the files of a private group one must join the group and then login to (mount) the group. Once the login processes is completed the group will appear as a virtual disk in the Document Explorer as groupname:// and it will also be available throughout the system.

Group actions, such as joining, leaving and creating groups are available in the Process XML language and the Shell Application. To facilitate this functionality will also be available in Document Explorer and in time other system applications. The applications utilize the Process XML functionality while the Shell Applications has special built in features as it is a component.

Listing Groups and subGroups

A list of of all available groups (hidden groups are not listed) is available to all users in the

Listing of all groups

home://Groups/

Creating Groups

By creating a group you reserve that specific group name and the creator becomes the owner of the group. The owner has full rights to the group, including creating additional user groups and to appoint users to "Administrators" which can utilize functionality that ordinary members cannot.

A group can be created in any of the following ways:

Using Process XML

ActionURL syntax:

```
[channel name]://createGroup([groupName], [owner alias])
```

Example:

```
<operation name="open" value="home://createGroup(Jennys Group , Jenny)"/>
```

Using Shell

To perform the same action in the Shell application you write one of the following commands:

```
groups -n "Jennys Group" Jenny
type createGroup(Jennys Group , Jenny)
```

Document Explorer

Open the Document Explorer application and select the Group folder. A list of groups should be displayed. If you right-click on a group icon a context-menu with option to that specific group will appear. If you right-click outside a group you should instead get a menu with the option "Create New Group"

Once you have chosen that option a new window will appear notifying you of the need to select a name for your group and your desired username (alias) in the group. Observe that you may not take a group name that has already been chosen.

If you try to register a group that already exists a system dialog will appear notifying you that the intended registration failed because it already exists.

If the group name is available and your registration was successful a system dialog should appear stating that the group was created. Your group is now available at the last group in the Groups folder.

Creating subGroups

The default setting for groups is that they support subgroups. This means that groups can be created within those groups and are considered to be subGroups. Creation of subgroups can be denied by the group owner. In order to create a subgroup the parent group (top level group) must be mounted.

Subgroups are created by any of the following ways:

Using Process XML

ActionURL syntax:

```
[channel name]://createGroup([groupName], [owner alias])
```

Example:

```
<operation name="open" value="xidezone://createGroup(Jennys Group , Jenny)"/>
```

Using Shell

To perform the same action in the Shell application you write one of the following commands:

```
xidezone://> groups -n "Jennys Group" Jenny xidezone
xidezone://> type createGroup(Jennys Group , Jenny)
```

Using Document Explorer

This action is currently not supported in Document Explorer.

Joining Groups

Groups can be joined by any of the following ways:

Using Process XML

ActionURL syntax:

```
[channel name]://join([group Path], [user alias])
```

Example:

```
<operation name="open" value="home://join(Jennys Group, Kirk Smith)"/>
```

Using Shell

```
type join(Jennys Group, Kirk Smith)
groups -j "Jennys Group" "Kirk Smith"
```

Using Document Explorer

In the Groups folder right-click on a group and select the "Join Group" option from the context-menu

Leaving Groups

Members can choose to leave a group at any time. Observe that group owners can't leave their groups.

Using Process XML

ActionURL syntax:

```
[channel name]://leaveGroup([group Path])
```

Example:

```
<operation name="open" value="home://leaveGroup(Jennys Group)"/>
```

Using Shell

```
groups -q "Jennys Group"
type home://leaveGroup(Jennys Group)
```

Using Document Explorer

Right-click on group name and select the option "Leave Group", Observe that this option is only available if the you have not mounted the group.

Mounting Groups

Users can login to (mount) groups by any of the following ways:

Using Process XML

ActionURL syntax:

```
<operation name="filesystem">
  <mount name="finance">
    <group name="Finance"/>
  </mount>
</operation>
```

The example will mount the group Finance as finance://.

Using Shell

```
mount [virtual drive name] [group name]
mount finance Finance
groups -l [virtual drive name] [group name]
groups -l finance Finance
```

Using Document Explorer

In the Groups folder right-click on a group and select the "Login to Group" option from the context-menu.

Group members

For each user that joins a group a profile will be created inside the "Profiles" directory inside the groups shared virtual disk. The file is named after the users alias and does not have any file extension. It is however an XML file and does only contain a self-enclosing profile-element.

The type of information should be stored in these files is public information about the user, the reason being that the file is viewable by all members of the group. Non-public information should be stored on the users local file system (home://) where only the user can access the information.

Observe that members can change their alias in a group at any time by using the following command in the Shell application.

```
[channel]://> groups -h [new alias]
```

Estimating the number of members

To estimate the number of members of a group you need to do a filelisting of file files inside the Profiles directory. While only a maximum of 100 files will be listed in the atom feed, the total number of members can be found in the os:totalResults element at the top of the feed.

A list of all members in a group can be acquired by using the following command in the Shell Application

```
groups -m "Jennys Group"
type home://Groups/Jennys Group/Members
```

Customizing Group Behavior

There are a few ways to alter or customize the behavior of groups. These include auto mounting and auto starting applications.

Auto mounting

To make a group automount it needs to be defined in the #Settings. Note that this will only work if the group has been mounted once before, and therefore created a channel node in #Settings. All groups can be found in the "channels" element as single channel elements.

```
<operation name="change" value="#Settings">
  <store type="replaceText" select="/settings[1]/channels[1]/channel[@path='Jennys Group']/@automount">true</store>
</operation>
```

Auto starting applications

Groups can also launch applications when mounted. These can then utilize the groups shared virtual disk. This is possible by using an autostart.xml file which is placed in the groups root folder.

The autostart.xml is an process XML file that might look like this:

```
<?xml version="1.0"?>
<process name="jennysAutostart" xmlns:xpc="http://xcerion.com/process/1.0" xpc:ns="http://xcerion.com/process/1.0">
  <step id="1" name="Open Application">
    <operation name="open" value="apps/JennyApp.xml"/>
  </step>
</process>
```

The code above will simply open a file located at "apps/JennyApp.xml". Observe that since this application uses the shared virtual disk it does require that the user has mounted the disk in order to work. If the group is not mounted the application will not appear correctly.

Role handling

All commands for role handling are currently handled through the Shell application. If the channel is xios then a group name must be provided. If the channel is the group then no group name should be provided.

Listing roles

Lists all available roles in a group. Excluding owner group and admin group.

```
groups [group name] -o
```

Creating roles

Creates a new role for the group.

```
groups [group name] -c [role name]
```

Adding role members

Promotes the member to a certain group. Note that the user id must be known.

```
groups [group name] -a [role name] [user id]
```

Removing role members

Demotes the member from a certain group. Note that the user id must be known.

```
groups [group name] -r [role name] [user id]
```

Group profile.xml

Every group administrator can create a profile.xml document that can be placed in the root folder of the group. This document will contain information about the group and will saved treated in a special way by the server, because it is indexed for the purpose of retrieving information about a group using the "Properties" menu option.

```
<?xml version="1.0"?>
<groupDescription xmlns:xpc="http://xcerion.com/process/1.0" xpc:ns="http://xcerion.com/profiles/group/1.0">
  <name>GroupName</name>
  <owner>Alias of group owner</owner>
  <icon>http://sport.com/xios/pics/icon.ico</icon>
  <logo>http://sport.com/xios/pics/logo100x100.gif</logo>
  <terms> The terms for joining the group</terms>
  <shortDescription>
    A description is that is less than 100 characters...
  </shortDescription>
  <fullDescription>
    A full description of the group.
  </fullDescription>
  <keywords>
    <keyword>keyword1</keyword>
    <keyword>keyword2</keyword>
  </keywords>
  <userProfileTemplate>
    <data id="userProfileField_name" caption="Name" component="input" indexed="true" required="true" value=""/>
    <data id="userProfileField_city" caption="City" component="input" indexed="true" required="false" value=""/>
    <data id="userProfileField_profession" caption="Profession" component="input" indexed="true" required="false" value=""/>
  </userProfileTemplate>
</groupDescription>
```

Example: profile.xml document syntax

Managing File Types

File types are used to associate certain files to certain applications and the ability to apply custom look and graphics for that file type. They are also used to default XIOS/3 applications such as Document Explorer, Document Selector and File selection dialogs. All file types are stored in every users settings file and can be modified.

Creating file types

To create a file type XML data must be appended to the settings.xml, which is available as #Settings. In order to this we use the change operation with the append type. We add a "type" element consisting of the two required attributes and two optional attribute, namespace and element. You can read about their purpose in the "File Type Association" section.

We append a description child element that is used as a longer description of the file type and can be used for tool-tips. The open, edit and preview element are used to define which application should be used to open the file, which should be used to edit the file (for XML files the XML editor might be a good choice) and which file should be used to preview files.

The icon element contains an URL to the icon in 32x32 format. Read more about this in the Existing File Types section. The listview element is used to define additional keywords which might be used to display different listviews depending of the content of this element.

```
<operation name="change" value="#TypeManager">
  <store select="/types" type="append">
    <type name="testXML" mime="text/xml" namespace="http://xcerion.com/doc/test" element="testElement">
      <description>TEST XML Document</description>
      <open>apps/desktop/noteapp.xml</open>
      <edit>xios/apps/XMLEditor/xmleditor.xml</edit>
      <preview></preview>
      <icon>gui/icons/32x32/document_xml.png</icon>
      <listviews></listviews>
    </type>
  </store>
</operation>
```

Editing file types

Using the file type created above we can again use the change operation to replace the text() node of the icon element with a new icon. So instead of document_xml.png it will be document_text.png. Observe that the Document Explorer might to be reloaded for the change to take effect. This is because XSLT files are not updated unless using the updateRedraw method.

```
<operation name="change" value="#TypeManager">
  <store select="/types/type[@name='testXML']/icon/text()" type="replaceText">gui/icons/32x32/document_txt.png</store>
</operation>
```

Deleting file types

Deleting files types is a matter of deleting the node-set from the #Settings XML document using the change operation.

```
<operation name="change" value="#TypeManager">
  <delete select="/types/type[@name='testXML']"/>
</operation>
```

Searching the Filesystem

This chapter is intended to explain how to perform searches and also how to filter searches. The documentation will show that the filesystem can be used a replacement of SQL based database.

It is possible to only retrieve the number of results found, without having the server send the information. This has advantages since is results in less bandwidth for the end-user, however the server load is the same.

Query Syntax

The query language that is used to communicate with the server is rather simple, yet if used correctly it can be very powerful. A search query is a list of search terms separated by white-space and is presented in any of the following ways:

1. [+/-][prefix:]term[*]
2. [+/-][prefix:]'term"term term'[*]
3. [+/-][prefix:]"term'term term"[*]

The information contained in the brackets is optional. The plus and minus sign in front of the prefix is used to define if the term must or must not exist. This is similar to "AND" and "AND NOT" in SQL. Unprefixed terms are neither plus or minus. They are instead used to increase the weight of a result and make it more relevant.

The prefix is also optional, but highly recommended to use. It consists of the local-name of an element, meaning that the namespace should not be used. As we mentioned in the meta data section of "Managing Files" the local-name must be unique in meta data field else wise search will not be possible. If the prefix is omitted it will search all searchable meta data fields.

A term is often simply a word. If that is the case then apostrophes and quotation marks should not be used. If a phrase is used and it contains white space then these must be used to enclose the search phrase. Observe that if the phrase contains quotation marks then apostrophes should be used to enclose the term and vice versa.

To make the server return files the contains words that start with the term supplied we must append an asterisk to our term.

1. home://Documents/?+title:Acme*
2. home://Documents/?+summary:'Jenny"s Party'*
3. home://Documents/?+summary:"Jenny's Party"*

Note that we are using a slash at the end of the folder path to indicate that we want to deal with XML data. If the slash was omitted the search will fail.

Filtering

Filtering search results is done by adding one or more search conditions. These are separated by white space. Only search results that match all conditions will be returned.

    home://Documents/?+title:Acme -title:Sales -title:Jenny

Special handling of atom:published, dc:startdate and dc:enddate

These meta data fields are reserved and will be handled in a special way. The atom:published which is set automatically by the server is not indexed. Instead three separate values are indexed from the timestamp. The following date "2019-02-04" would index the following values:

pubyear:2019

pubmonth:02

pubweek:06

A search query to fetch all files created in February 2019 might look like this:

type home://Documents/?pubyear:2019 pubdate:02

The dc:startdate and dc:enddate which might be used to create appointments which span over a time period and not just a moment in time might look like this when added as meta data:

```
<dc:startdate date="2018-12-10" time="01:54:00" zone="Z"/>
<dc:enddate date="2019-01-09" time="00:00:00" zone="Z"/>
```

It would result in a special indexing that would save the following data

year:2018

year:2019

month:12

month:01

week:50, week:51, week:52, week:01, week:02

No search results

When the server does not find any files matching the search criteria it will only return the following XML. Observe that this is the same as when there are no files in a folder.

```
<?xml version="1.0"?>
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom" xmlns:os="http://a9.com/-/spec/opensearch/1.1/"
        xmlns:dc="http://xcerion.com/directory.xsd" xmlns:ni="http://xcerion.com/noindex.xsd">
  <os:totalResults>0</os:totalResults>
  <os:startIndex>0</os:startIndex>
  <os:itemsPerPage>100</os:itemsPerPage>
</atom:feed>
```

Development Tools

This section of the documentation contains information about the various XIOS/3 developer tools. Currently the section is very brief but will in time be expanded to contain external tools, third party extensions and user developed applications.

XIOS Applications

To aid developers in their development of applications XIOS/3 has developed an XML editor and a Visual IDE that will be available for all XIOS/3 users.

XML Editor

XIOS/3 general purpose XML editor offers basic XML editing. The functionality offered here is tools needed for editing XML documents. The application will check if the document is well formed, display line numbers as well as search and replace functionality.

The state of the automatic updates feature can be set to off which allows the user to edit documents such as #Settings but not update the document until the users requests update. Observe that this behavior differs from the save action which can be performed at any time. More features will be added in the future.

This application is still in development and not ready, but you can try it using the following command in Shell:

load "xios/apps/XMLEditor/xmleditorapp.xml"

This application uses the XMLEditor component which allows anyone to make their own XML editor.

Icons and Images

When developing applications you will most likely make use of images or icons for your applications. For this you can use either the icons provided by XIOS/3 or use your own icons.

```
<image src="xios/icons/applications/16x16/about.png" width="16" height="16"/>
```

Example: Base server icon path

If no protocol is given, the icons will be loaded from the base server (where the top HTML page is hosted). In all other cases the icon will be loaded through the XIOS/3 filesystem.

```
<image src="home://Documents/image.png" width="16" height="16"/>
```

Example: XIOS/3 file system icon path

Application Packaging

Application Packages are XML files that are used to create a virtual package, in the form of a manifest file, for all files used in your application. This file will allow you to run multiple instances of your application and enjoy the advantages of application isolation.

The application file also allows for limiting the number of active application instances by a single end-user.

For small applications, with little code or very few files use inline application packages. When dealing with larger application when use regular application packages. Both of these utlize application package files. However with inline application packages all the code in written directly in one single file. When dealing with regular application packages you have an application file which only makes references to the view and process files.

Storage

Application can be stored in your file XIOS/3 filesystem (home://). The applications are executed as naitive XIOS/3 software. To facilitate the readability of your application there is a file structure that should be used which divides the different parts of the applications in to separate folders. An application should have the following folders:

- views
- processes
- data
- layout
- images

The root folder should contain the application file. In the views and processes folder all UI XML and Process XML files should be placed. The data folder contains static data that the end-user cannot alter. The layout folder is a multi-purpose folder containing CSS files and XSLT files used in XSLT Driven Components. Finally the images folder contains all images of any sort. Again this structure is for increased readability of your application. Also remember to declare which files must be downloaded when the application goes in to offline-mode.

Overall Structure

There is also the possibility of writing the view and process code directly inline inside the application file. This way of developing has great advantages when writing small apps or widgets and will reduce the number of server requests and thus make your application load faster.

An additional feature gained from using the manifest file is the possibility is use the application in a offline-mode. This is a planned feature; but applications can be adapted today.

Explaining Xlink

In the application package concept xlink is frequently used. Xlink is a W3C standard that allows you create and describe links to other XML documents. This is very important as it make it possible to load file when they are needed making your applications faster.

Every xlink document has three attributes; xlink:type, xlink:actuate and xlink:href. They all use the xlink-namespace which is the following URI http://www.w3.org/1999/xlink. the xlink:type attribute can be set as either simple or extended. In XIOS/3 only xlink:type="simple" is used.

The xlink:actuate attribute is also of the type enumeration and can be set to either onRequest or onLoad. Controls when the link will be executed. xlink:actuate="onLoad" means that the link/file will load when the application package is loaded while xlink:actuate="onRequest" means that it is loaded when the file is requested within the application.

NOTE: The onLoad option does not trigger the Opened events for views.

The final attribute xlink:href is the URL of the file that you want to load.

Element: application

The application element is the root element of application package files.

Required attributes

Name	Type	Values	Description
name	string	unique name	The name of the application.
icon	string	URL	The icon of the application
instances	integer	**0**, [1...n]	The number of instances that should be allowed
xmlns:xlink	string	http://www.w3.org/1999/xlink	The URI of the xlink namespace

Element: share

If the application should be abled to be sharable across user's. Meaning if the Share Window feature (GBC - Gesture Based Collaboration) can be used for the application.

Since Window Sharing is enabled by default, you don't need to define it. However in order to disable the feature you need to set the text() node value to "false". This might be needed in some application that are intended for a single user, deals with sentitive information or where the results depends on which user is viewing the application (ie #Friends data document will differ from user to user).

```
<application...>
  <share>false</share>
...
```

Example: Disabling window sharing

Element: settings

Here we define the application settings.

Required attributes

None available.

Child Element: path

Contains information about from which virtual drive the application will be executed. Usually this is home://, however when using groups the path might be the groups virtual drive.

Child Element: home

Can contain the URL to the XIOS/3 folder where tile files of the application are stored. The contents of this file are available as the system alias $home when defined. The result will be a file listing of the folder contents.

Child Element: localedir

The location of the folder which contains the localization files, allowing the application to be translated to another language.

Element: about

Here we define information about the application that will be displayed inside the control panel.

Required attributes

Name	Type	Values	Description

loader	boolean	false, true	Decides is a loader should be shown while the application is loading
version	numerical	1.0	The version of the about element.
revised	date	YYY-MM-DD	The date when the application was last revised
created	date	YYY-MM-DD	The date when the application was created
author	string		The author of the application
copyright	string	Any string value	The copyright holder of the application
publisher	string	Any string value	The publisher of the application

Child Element: description

Contains a descriptive information about the application.

Child Element: license

Contains the license under which the application is released.

Required attributes

Name	Type	Values	Description
xlink:type	enumeration	simple	The type link the URL is
xlink:actuate	enumeration	onRequest, onLoad	Controls when the link will be executed
xlink:href	string	URL	The URL of the license

Child Element: installed

Will contain a date when the application was installed.

Element: resources

In this element we need to define all resources that are needed to make the application run offline. Links to images and xsl-files supplied in the UI XML files will be downloaded automatically. The XSL file is available as "#nameDefined" and is to be defined in the href attribute of the component that utilizes xslt to render XML.

Child Element: item

This element can be used to declare inline XSL files which is made possible by defining a name attribute and then the XSLT as child elements of the item element. If no name attribute is supplied then the xlink attributes are needed and are used to define the information needed to download the resource for offline use.

Required attributes

Name	Type	Values	Description
name	string	Any string value	The name of the item
xlink:type	enumeration	simple	The type link the URL is
xlink:actuate	enumeration	onRequest, onLoad	Controls when the link will be executed
xlink:href	string	URL	The URL of the license

Element: view

Every view file should be defines as a child element of the application root element.

Required attributes

Name	Type	Values	Description
xlink:type	enumeration	simple	The type link the URL is.
xlink:actuate	enumeration	onRequest, onLoad	Controls when the link will be executed.
xlink:href	URL		The URL of the license.

Element: process

Like view files, every process file should be defined as a child element of the application root element.

Observe that when declaring the main process file the xlink:actuate should be set as onLoad so that the file will be opened when the application is launched.

Required attributes

Name	Type	Values	Description
step	string		The step that should be initiated.
xlink:type	enumeration	simple	The type link the URL is.
xlink:actuate	enumeration	onRequest, onLoad	Controls when the link will be executed.
xlink:href	URL		The URL of the license.

Regular Package

This is how an application file might look like. It is very simple and as mentioned earlier it is a definition of your application.

```xml
<?xml version="1.0"?>

<application name="HelloWorld" icon="icon://earth" xmlns:xlink="http://www.w3.org/1999/xlink">
  <share>false</share>
  <settings>
    <path>home://</path>
  </settings>

  <about loader="false" version="1.0" revised="2019-04-10" created="2019-02-05" author="XIOS/3" publisher="XIOS/3">
    <description>This is a test application</description>
    <license xlink:actuate="onRequest" xlink:href="https://xios3.com/license/lgpl_2.1.xml">GNU LGPL 2.1</license>
    <installed>2020-03-10</installed>
```

```
13   </about>
14
15   <resources>
16     <item xlink:actuate="onRequest" xlink:href="home://Documents/countries.xml"/>
17   </resources>
18
19   <view name="HelloWorldProcess" xlink:actuate="onLoad" xlink:href="home://Application/HelloWorld/views/mainUI.xml"/>
20   <process step="1" xlink:actuate="onLoad" xlink:href="home://Application/HelloWorld/processes/mainProcess.xml"/>
21 </application>
```

Example: Application Package for HelloWorld

Line 04

Here we define if the application should support Gesture Based Collaboration (GBC) or Application Sharing as it is publically known.

Line 05-07

Here the settings element is declared as well as the path child element. As this is a regular application we will set the path to home:// indicated that the application should be run on your local file system.

Line 09-13

The about section contains all required attributes. The loader is set to false since our application will load fast; thus no one is needed.

We will also set the author, copyright and publisher attributes to XIOS/3.

The description child element contains a short line description about the application.

We will also set the license element to the Xfree license. Finally the installed element will just contain a date.

Line 15-17

In the resources section we define the one file we would need to download is the application goes offline.

We define one item child element to initiate onRequest. This file is a XML data file containing information about countries.

Line 19-20

The final two child element will link to our only view file and our only process file.

We will set the process file to onLoad to ensure that it starts when the application starts. We will also set the step attribute to 1 which is our first step that will open the view file.

Inline Package

This is an example of an inline application package. Here we define inline UI XML (View) and Process XML, as well as inline CSS and XSLT. A regular application might divide the functionality in to different files however there is always an option to write this code as inline. This has very positive results as fewer requests are made to the server and the application (or widget) will load faster.

```
 1 <?xml version="1.0"?>
 2
 3 <application name="packagetest" icon="xios/icons/network/32x32/earth.png" xmlns:xlink="http://www.w3.org/1999/xlink">
 4 <settings>
 5   <path>home://Applications/</path>
 6   <home>home://Documents/</home>
 7 </settings>
 8
 9 <about loader="false" version="1.0" revised="2019-04-10" created="2019-02-05" author="XIOS/3" publisher="XIOS/3">
10   <description>This is the application description of Hello World.</description>
11   <license xlink:type="simple" xlink:actuate="onRequest" xlink:href="https://xios3.com/license/xios_1.xml">XIOS.1</license>
12   <installed>2013-03-10</installed>
13 </about>
14
15 <resources>
16   <item name="customXSL.xsl">
17    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
18      <xsl:template match="//book">
19        <div class="boldHeading">Title: "<xsl:value-of select="@title"/>"</div>
20      </xsl:template>
21    </xsl:stylesheet>
22   </item>
23   <item name="myDataDoc">
24     <books>
25       <book name="1985" author="George Orwell"/>
26     </books>
27   </item>
28 </resources>
29
30 <view xlink:actuate="onLoad" name="packagetest" title="Inline Application Package" author="XIOS/3" version="1.0" height="400" width="400"
31       resize="true" icon="xios/icons/network/16x16/earth.png">
32   <style>
33   .boldHeading{
34     font-weight:bold;
35     font-family:arial, verdana;
36     font-size:16px;
37   }
38   </style>
39   <panel name="panelContainer" height="100%" width="100%" type="row" padding="5">
40     <panel name="mainPanel" type="column" width="100%" height="100%" padding="5">
41       <render name="renderContent" href="#customXSL.xsl" height="100%" width="100%" scroll="false"/>
42     </panel>
43   </panel>
44 </view>
45
46 <process xlink:actuate="onLoad" name="packagetest">
47   <step id="1" name="Iniate Application">
48     <operation name="bind" value="#myDataDoc">
49       <component view="packagetest" name="renderContent" select="/lists/list[1]"/>
50     </operation>
51   </step>
52 </process>
53 </application>
```

Example: Process code for inline application package. Here we have placed UI XML, Process XML, XSLT and CSS in to one file.

Line 01-08

In the first few lines we first have the XML processing instruction on line 1. Then the application root element on line 3. Observe that the xlink namespace is declared here. The reason for this is because xlink is used when dealing with view and process files.

The settings element contains two child elements: path and home.

Line 09-14

In the about element we describe information regarding the application. The information that is stored within the elements of this section is not mandatory to make the application work, however it is good practice to do so.

Line 15-29

The application package file allows you to write inline data and XSLT by declaring them as resources. These can then by used by different UI XML files, which are defined in the Application Package. The inline files are the same as external files, however they offer a great advantage when working with widgets. Declaring XSLT and data files inline will reduce the load time of you application allowing it to start faster. Observe that there are not xlink declarations on the item elements on the inline resources.

Our XSLT is very basic. First the root element, xsl:stylesheet and the xsl namespace declaration. We then proceed to match any book element in the data that we bind, and to print the value in the name attribute of that element. Note that we are using the class attribute on the HTML div element when we are outputting the book element's title attribute. This is made possible by the inline CSS declared in the view file below.

Line 30-44

In this part of the application package we describe our file. We start off with the view root element and the xlink:actuate which in our case must be set to onLoad given that it is our only view file and that we will not open it directly from the process code. The view element has some attribute that control the visual appearance of the view file, such as height, width and icon.

Next is the inline CSS which de describe in the style element. Here we only define one class, "boldHeading" which will be bold, in Arial font and will be 16px in height. After the style element we describe our regular UI XML with panels and our XSLT driven component, render. When we refer to the XSL file in the href attribute of the component we refer to #customXSL.xsl, which was the name that we defined for the inline XSL in the resources section of the application package.

Line 45-52

Here we define our process XML file, which consists of a process element with a xlink:actuate attribute for which the value should be set to onLoad. We will only use one step node-set where we will bind the data to the render component. Observe that the id attribute of the step element is set to 1, because the application will always start by opening step with the 1 (unless otherwise defined) and not the first step by order.

Widget

This is an example of a widget. It looks very similar to a regular application package, but here we make use of inline UI XML and Process XML code. As a result we get faster loading time because fewer requests are made to the server.

In order to attach widgets to the widgetpane they also need to be defined in #Settings (xpath: /settings/widgets).

```
 1  <?xml version="1.0"?>
 2
 3  <application name="HelloWorld" icon="xios/icons/network/32x32/earth.png" xmlns:xlink="http://www.w3.org/1999/xlink" instances="1">
 4  <about loader="false" version="1.0" revised="2019-04-10" created="2019-02-05" author="XIOS/3">
 5    <description>This is the application description of Hello World.</description>
 6    <license xlink:type="simple" xlink:actuate="onRequest" xlink:href="https://xios3.com/license/xios_1.xml">XIOS.1</license>
 7  </about>
 8
 9  <view xlink:actuate="onLoad" name="HelloWorldView" title="Hello World Widget" height="40" width="160" mode="widget">
10    <panel name="mainPanel" type="flow" height="auto" width="auto" padding="3">
11      <button name="buttonAlert" height="30" width="150"/>
12    </panel>
13  </view>
14
15  <process xlink:actuate="onLoad" name="HelloWorldProcess">
16    <trigger view="HelloWorldView" component="buttonAlert" event="Select" step="3"/>
17    <step id="1" name="Initiate Widget">
18      <operation name="call" value="2"/>
19    </step>
20
21    <step id="2" name="Set Button Text">
22      <operation name="callMethod" value="#HelloWorldView#buttonAlert">
23        <method name="setValue">
24          <param type="string">Click Me!</param>
25        </method>
26      </operation>
27    </step>
28
29    <step id="3" name="Component Clicked">
30      <operation name="callMethod" value="#HelloWorldView#buttonAlert">
31        <method name="setValue">
32          <param type="string">Hello World!</param>
33        </method>
34      </operation>
35      <operation name="action" value="#HelloWorldView">
36        <component name="buttonAlert" action="disable"/>
37      </operation>
38    </step>
39  </process>
40  </application>
```

Example: Application Package for HelloWorld Widget

Line 01-03

On the first line is the mandatory XML processing instruction with its attributes. The third line contains the root application element; with its attributes name icon and then the xlink namespace declaration. The latter is needed because xlink is used (read more in the beginning of this document). Here we have set the instances attribute to 1 effectively limiting the number of simultaneously to only 1.

Line 04-07

The about section contains all required attributes. The loader is set to false since our application will load fast; thus no one is needed.

We will also set the author, copyright and publisher attributes to XIOS/3.

The description child element contains a short line description about the application.

We will also set the license element to the Xfree license. Finally the installed element will just contain a date.

Line 09-13

Here is our inline UI XML code. It starts with the view element, which is the root element of view files. As you might see that this code is regular UI XML code. There difference is that we need to set the xlink:actuate attribute to onLoad, meaning that it will be loaded when the application is started. Also note that there is an attribute called mode on the view element. This is set to widget because we want the be able to access widget behavior.

The rest of the UI XML is pretty simple. The view has a name (required), a height (required), a width (required) and a title (required). It contains a panel which itself contains a button. The button is the center of our Hello World widget.

Line 15-16

This is where our inline process file starts. The xlink:actuate is also set to onLoad, and just like with the view file it means that it will load when the application package is loaded. The required name attribute is also set.

In this widget application we will only use one trigger element. It is set to listen a to our button (buttonAlert) in the HelloWorldView-view file. The event that will be expected is Select and when that is detected it will execute step 3.

Line 17-19

In our first step, which we should designate 1, as it is the default step to start, we will only perform one simple operation and that is to call step 2.

Line 21-27

This step is called 2 and the only purpose of this step is to set the text of the button using the callMethod operation and the setValue method. By setting the value of the operation to #HelloWorldView#buttonAlert we notify XIOS/3 that we want to execute the method on a component called buttonAlert in the HelloWorldView-view file. This step is called from step 1.

Line 29-38

Our third step, called simply 3, will be executed when our button is clicked. As you might recall we used a trigger element to catch the Select event when that has been made by the button. When that occurs this step is executed.

The step starts by using callMethod to change the text from Click Me! which we set in step 2, a another string value, Hello World!

Finally we use an action operation to disable the button component. Thus making it disabled and grayed out.

Line 39-40

Here we just close the process element and the application element.

Extending XIOS/3

XIOS/3 is designed to be dynamically extended with new components, operations, protocol, and more. Since both views and processes are abstracted it means the underlying implementations can not only be dynamically loaded on demand but also replaced within a running XIOS/3 instance.

Additional Components

The most complex part of XIOS/3 to extend is the component library, since it typically involves JavaScript code, XSL template, and dependencies. All of these aspects are described for built in components in the system configuration document which by default is loaded from xios/config/system.xml. It is possible to modify this document or create a new configuration file and load instead. Beware of the XIOS/3 preload system that bundles resources in the system file, intercepting attempts to load a modified system.xml unless you remove or replace it. To instead point to a different resource, just add the "system" property to the boot object.

```
<script>
  var boot = { system : "/system.xml" };
</script>
```

Example: Bootloader parameters

A simpler way to just add a few components to the system configuration is to use a script tag of type "xios" and declare the name attribute "addConfig" with the components to add. As a simple example we can use this XSL only component that defines a loader spinner. The XSL code creates a rotating, animated waiting spinner with configurable size and color.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:template match="spinner">
    <xsl:variable name="width">
      <xsl:choose>
        <xsl:when test="@width"><xsl:value-of select="@width"/>px</xsl:when>
        <xsl:when test="@height"><xsl:value-of select="@height"/>px</xsl:when>
        <xsl:otherwise>100px</xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="color">
      <xsl:choose>
        <xsl:when test="@color"><xsl:value-of select="@color"/></xsl:when>
        <xsl:otherwise>#000</xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <div id="{@name}" style="width: {$width};">
      <svg class="circular" viewBox="25 25 50 50">
        <style>
@keyframes
rotate {  100% {
    transform: rotate(360deg);
  }
}

.circular {
  animation: rotate 2s linear infinite;
  transform-origin: center center;
}

.path {
  stroke-dasharray: 89, 200;
  stroke-linecap: round;
  stroke-width: 2;
  stroke-miterlimit: 10;
  stroke: <xsl:value-of select="$color"/>;
}
        </style>
        <circle class="path" cx="50" cy="50" r="20" fill="none"/>
      </svg>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

Example: spinner.xsl

The component is then added to the set of components using the following addition to the component definitions

```
<script type="xios" name="addConfig">
  <component name="Spinner" tag="spinner">
    <skin href="xios/ui/Spinner/spinner.xsl"/>
  </component>
</script>
```

Example: HTML script tag

For a more complicated example that involves JavaScript code as well as dependencies on external resources, let's look at a toggle from Fluent UI implemented with React. The components are implemented in a single resource called office-ui-fabric-react which, as the name implies, depends on React. This is actually two dependencies, one on React DOM which in turn depends on React. This dependency graph is show in this component definiton.

```
<script type="xios" name="addConfig">
  <component name="FluentToggle" class="CA_FluentToggle" href="/Fluent/CA_FluentToggle.js" tag="fluent-toggle">
    <dependency href="https://unpkg.com/office-ui-fabric-react@7/dist/office-ui-fabric-react.js" symbol="Fabric">
      <dependency href="https://unpkg.com/react-dom@16.8.6/umd/react-dom.development.js" symbol="ReactDOM">
        <dependency href="https://unpkg.com/react@16.8.6/umd/react.development.js" symbol="React"/>
      </dependency>
    </dependency>
    <skin href="/Fluent/fluent.xsl"/>
  </component>
</script>
```

The XSL in this case is trivial, a single div element to hand over to React for it to render its DOM into.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:template match="fluent-toggle">
    <div id="{@name}"/>
  </xsl:template>

</xsl:stylesheet>
```

The JavaScript is where all the relations between XIOS/3 internals and the API of the component are established. First of all on line 1 the entire code is wrapped in an anonymous function to not leak any symbols into the global variable space. Instead on line 73 there is a registration call using the class name used inthe component definition. Then line 2 actually defines the class, extending the generic XIOS/3 Component class. The first entry point for this component wrapper is the init method where a dependency object is given as argument. This object maps from the symbol names in the component definition to the actual loaded objects.

The actual code in the init method of course varies depending on the component to wrap, but the common pattern for React comopnents is to define a class that extends the React.Component class (line 10). In it the state is set up according to parameters from the view XML code (line 14-18), callback for changes (line 17) and a mechanism to make the React object externally visible for state changes (line 19).

All the rest of the methods, with the exception of toggle, are Component API methods. unload is called when the component is permanently removed. setValue and getValue are used to see and alter the value of the component directly. update is used if the component is bound to another data source, like a document, and updates to the state is signalled through this method. enable and disable are called from the respective action operations.

```
 1  !function()
 2
 3  class CA_Fluent extends Component {
 4
 5    init(dep) {
 6
 7      const _this = this;
 8      this.reactdom = dep.ReactDOM;
 9
10      class Toggle extends dep.React.Component {
11        constructor() {
12          super();
13          this.state = {
14            disabled : _this.ui.getAttribute("enabled")==="false",
15            onText : _this.ui.getAttribute("onText"),
16            offText : _this.ui.getAttribute("offText"),
17            onChange : _this.toggle.bind(_this),
18          };
19          _this.obj = this;
20        }
21
22        render() {
23          return dep.React.createElement(dep.Fabric.Toggle, this.state);
24        }
25      }
26
27      this.reactdom.render(dep.React.createElement(Toggle), this.getElement());
28    }
29
30    unload() {
31      this.reactdom.unmountComponentAtNode(this.getElement());
32    }
33
34    getValue() {
35      return this.checked ? "true" : "false";
36    }
37
38    setValue(to, winEvt) {
39      to = (to === "true");
40      this.obj.setState({checked:to});
41      this.toggle(winEvt, to);
42    }
43
44    update(evtObj) {
45      if( !evtObj || evtObj.ruleUpdate ||
46          (evtObj.updates && evtObj.updates[0].originator !== this) ) {
47
48        if( this.xlink &&
49            (this.dataNode = this.xlink.selectSingleNode(this.xpath)) ) {
50          this.setValue(this.dataNode.textContent);
51        }
52      }
53    }
54
55    toggle(winEvt, to) {
56      if( to == this.checked ) {
57        return;
58      }
59      this.checked = to;
60      sys.evtMng.notifyListeners(this, to ? "Select" : "Release", winEvt, null);
61      this.setProperty("selected", to ? "true" : "false", winEvt, true);
62    }
63
64    enable() {
65      this.obj.setState({disabled:false});
```

```
  }

  disable() {
    this.obj.setState({disabled:true});
  }
}

boot.addComponent("CA_Fluent", CA_Fluent);

}();
```

Example: CA_FluentToggle.js

Additional Operations

The individual operations called from the process langauge in XIOS/3 are quite simple. All the complicated state management, resource management, expression handling etc. are handled by the calling layer above the operation implementations. Additional operations can be added either by setting boot.operations to an object mapping operation name to a corresponding function, or by putting the operation code in a separate javascript file and declare it a dependency in the boot depend section.

```
var boot {
  depend : {
    js : [ "/ops/PO_validate.js" ],
  }
};
```

Example: Bootloader parameters

The code of the operation is simply an asynchronous function that returns a promise that resolves when the operation processing is done. The operation function is called with a process object representing the current running process, a context object representing the context in which the process is operating, an XML element node corresponding to the operation node in the process XML document, and an event object representing the event triggernig this operation.

```
!function() {
  boot.addOperation("validate", PO_validate);
  function PO_validate(process, context, operation, evt) {
    // Return Promise
  }
}();
```

Example: PO_validate.js

UI Development

In this section we will cover how to build a UI for an application, the fundamentals in UI layout concepts in XIOS/3 and how visual components are put together.

UI XML is the visual part of XIOS/3 Application Development. When considering the MVC design pattern, UI XML equals the view.

What this section covers

This part of the documentation will cover all parts of UI Development. The component attributes section will explain everything you need to know about attributes and how they affect components. Here we also cover the concept of required and optional attributes as well as what type of attribute values are possible to set.

UI XML also gives the developer a possibility to change components states by using component actions. Which are executed inside the application logic (Process XML). There is also the possibility to apply methods on components and that is also covered in this section. By using methods and actions you can change how the end-user interacts with the GUI environment.

As the main part of UI XML the components are the center of attention in this section. The section page Component Types explains the types of components that exist.

We will also cover Rules and explain how these can be used to dynamically alter components depending on what data is bound to the component.

This part of the documentation is often updated so remember to come back to see available attributes, events, methods and actions. This is always kept up to date.

Overall Structure

This section covers the view element explain how to use its attributes as well as the difference between widgets and normal applications. We also briefly explain how toolbars and menubars are used.

The final section titled layout covers the basics of Application Layout and some optimization tips.

View

The view element is the root element for all view-files (UI XML files).

Usage

The most important attribute is name which is used when catching events from components located inside the view file. You can read more about this in the Expressions section.

In this element we also define if the view is a widget by setting the value inside the mode attribute. While widgets and regular applications are built in the same way, there are differences in how they are displayed when booted. Widgets lack a title bar and thus buttons for closing, minimizing and maximizing it. It also has transparent backgrounds which regular applications cannot have. Widgets have a special widget context menu that is the same for all widgets. Developers can add their own submenu within this context menu by adding a menu component. It will be picked up by the system and displayed as a submenu called options on the default context menu. Observe that you can only use a single menu component for this purpose.

A splash screen that is while the application is loading, will be displayed when the splash attribute is set to true. In this splash screen information contained whithin the title, copyright, publisher and created attributes.

This element offers many options when it comes to size. The height and width attributes are the starting size of the view. The view will be resizable by default but can be constrained by setting resize to false. Furthermore there are also four attributes which are used to control the maximum and minimum height and width of the view. If resize if set to false these will have no effect. The autoresize attribute which is available for widgets allows your widget to be resized according to the content inside the widget.

The icon attribute sets the application icon which is not visible if the mode attribute has been set to widget.

Attributes

Required Attributes

Name	Type	Values	Description
width	measure values	integer (px)	Start width of the dialog.
height	measure values	integer (px)	Start height of the dialog.
icon	string	URL	Path to dialog icon. This is shown in the title bar of the dialog when styled as an application window.
look	enumeration	**blue**, graphite, liquid, alert	Defines the style of the dialog when shown as an application window.
mode	enumeration	**main**, widget, window	Application mode to be used
name	string	Any string value	The unique name of the view
resize	boolean	**true**, false	If the dialog should be resizable

Optional Attributes

Name	Type	Values	Description
alwaysontop	boolean	**false**, true	If the dialog should stay on top (Highest Z-index)
author	string	Any string value	The author of the view
autoresize	boolean	**false**, true	Autoresize widget to fit content
bottom	integer	pixels	Position the dialog this many pixels from the bottom of the window. Overrides "top" attribute.
bottomExclusion	integer	pixels	If this view is put as the main view (background) this number of pixels at the bottom will be off limits to place a window title bar in.
close	boolean	**true**, false	Determines if the dialog is allowed to be closed.
copyright	string	Any string value	Copyright notice
created	date	[YYYY-MM-DD]	Creation date of view
draggable	boolean	**true**, false	Determines if the dialog is allowed to be repositioned.
handleClose	boolean	**false**, true	If closing of the application should be handled by the application and not by the system
iconbar	boolean	**true**, false	Defines if the icon in the titlebar should be displayed
left	integer	pixels	Position the dialog this many pixels from the left border of the window.
localedir	string	URL	The path to the folder which contains language localization files for translation
logotype	string	URL	splash screen logo (110x25px)
maxHeight	measure values	integer (px)	Maximal height of the view
maxWidth	measure values	integer (px)	Maximal width of the view
minHeight	measure values	integer (px)	Minimal height of the view
minWidth	measure values	integer (px)	Minimal width of the view
modal	boolean	**false**, true	Require user interaction to close window
position	enumeration	**default**, center	The starting location of the wiew
publisher	string	Any string value	Name of publisher
revised	date	[YYYY-MM-DD]	Last revision date of view
right	integer	pixels	Position the dialog this many pixels from the right border of the window. Overrides "left" attribute.
share	boolean	**true**, false	Enable/disable window sharing, if disabled the window cannot be shared to another user
splash	boolean	**false**, true	If a splash screen should be displayed while loading
taskbar	boolean	**false**, true	if a taskbar should be displayed
title	string	Any string value	The header/title of the view. Defaults to the view name.
top	integer	pixels	Position the dialog this many pixels from the top of the window.
topExclusion	integer	pixels	If this view is put as the main view (background) this number of pixels at the top will be off limits to place a window title bar in.
version	enumeration	**1.0**	File version (personal use)
winstate	boolean	**true**, false	If XIOS/3 should remember the position and size of the applicaiton

Events

These events are specific for views:

View Events

Methods

These methods only apply to view files:

attach, detach, executeScript, focus, maximize, minimize, getDimension, restore, restoreAppDlg, setImage, setDimension, setIcon, setTitle

Toolbars

The toolbar is a menu component that can be placed outside the wrapping panel.

Usage

Toolbars can be placed at the top, bottom, right or left of the main content. It can also be floatable. This menu type can use graphics and contain other components such as combobox and slider.

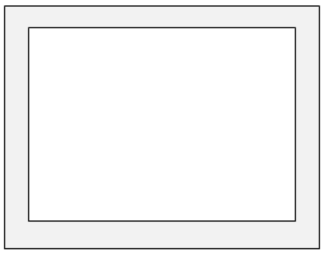


Figure: Toolbar placement area as represented by the grey colored (outer) area.

Menubars

The menubar is a menu component that can be placed outside the wrapping panel.

Usage

Menubars can be placed at the top, bottom, right or left of the main content. Unlike the toolbar this component is only text based and can only contain menu components.

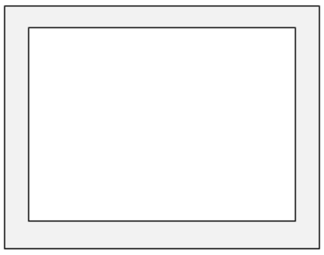


Figure: Menubar placement area as represented by the grey colored (outer) area.

Layout

Layout is one of the most important areas of XIOS/3 Application Development. It consists of three main parts; Usability, Optimization and Application Layout. Here we will cover the latter two.

Application Layout

Every UI XML file has a root element called view in which all the UI XML is stored. Inside this view element all immediately visible component code must be wrapped in a a Panel component, or a component servince the same role. Exceptions are the non-freely positioned MenuBar and ToolBar with their respective children, and elements not directly locked to the application window, like context menues.

The main purpose of the Panel component is to act as a container of other components. This means that it is used to form complex layouts. Panels can be placed inside each other to form desired layouts.

Basic Layouts

You can find a selection of basic layout examples and in the examples section.

Basic component Layouts

Almost all components that can be placed inside a panel (see Parent Element for each component) will behave like a panel. Meaning that they can be aligned, placed in rows and placed in columns.

Examples of component layouts can be found in the Applications Examples

Optimization

No matter if you are designing a small or large application, optimization is always important. When it comes to layout much can be done (or avoided) to make your application faster, better and more user-friendly.

You must learn to use Panels in a smart way. By that we mean that you should use as few as possible. When setting the width and height on panels (or on any other component for that matter) try to only use fixed values (pixels) rather than percentage or auto. That will make your application load faster.

Using Components

This section covers everything related to components. First the types of components that are available, followed by a section explaining how attributes are used and what values they can have. We then move on to list all available events, actions and methods that can be used on components or that emitted by components.

Here we also have a section entitled "Look and Feel" which covers the usage of CSS within XSLT files. We do not cover what CSS is or how that language is used but instead how to use it when developing XIOS/3 applications. The definition of rules and which components that can utilize them is also covered here.

A complete component reference is also featured which in great detail lists all attributes, available methods and events and much more.

Types

The UI XML language is based on components that are written as XML elements, with attributes and child elements to control them. There are two basic element types which the UI XML consists of; the High Level Element and the Child Element.

High Level Elements do not require any special type of parent component except for the mandatory Panel component. The reason for this is that the XIOS/3 rendering engine requires a Panel to be the only direct child element of the root element view. Child Element components are components that require a specific parent component other than the Panel element, an example of child element component is the DataItem which requires Menu as a parent element.

There are also several component types. These categorize the main purpose of the component and/or functionality. Some components are categorized as two or more of the types.

Application Components

Components that are extremely self-sufficient and are actually applications in themselves, but can be integrated in to you applications.

Atomic Components

Components that are very easy to use component that offer easy interaction to end-users. These components normally do not need bound data to work. Because of this they have limitations of what can be achieved by using them.

Data Driven Components

Components that need XML data to be bound to them to be able to function. Data driven components are the strength of XIOS/3, since you can bind components to data or each other (and if data is available share it). The data itself can come from either XIOS/3 or any web service on the web that generates XML.

Menu Components

Components that are used to construct a variety of different types of menus. There are several options and include expander menus, context menus (created when an end-user right-clicks on a component) and of course traditional toolbar menus.

XSLT Driven Components

Components that need XSLT files in order to function. These components also require data to be bound to them in order to render the XML data in a desired way. XSLT driven components offer much flexibility when designing applications, since they allow for a total customization of the data displayed as well as how it will be displayed.

Wrapper Components (Container Components)

Component that are caused to contain other component, either for layout purposes or for markup purpose to allow the system to handle child element component is a certain way using Rules for instance.

Attributes

All components contain a number of attributes controlling how the component should be initiated, interact and be visualized. How an attribute will affect the component will depend on what value it is set to. An attribute value could be of any of the types defined in section Attribute values.

A component must contain all the required attributes and may contain any or all of the optional attributes . To be able to control and customize specific behavior for components, a component can have its own specific attributes.

The component-specific attributes are described in section Component Reference, where all the components are explained in detail.

Attribute values

string

A string value is text that must start with a letter [A - Z | a - z] and may be followed by any number of letters, digits [0 - 9], hyphens [-], underscores [_], periods [.] and colons [:].

Example

Boston, Figure9, bookCase_19

boolean

A boolean value represents the value true or false. If any other value is specified the value will be interpreted as a false value. If an attribute of boolean type is expected but not present, it will be interpreted as a false attribute value.

Example

true, false

numerical

A numerical value is a digital number representing a negative - [0 - n] or positive [0 - n] integer value. The numerical value can be used to specify a width, height, length, iteration, interval or any other instance where numbers are needed.

Example

520, -95, 233

measure values

If the numerical value is a measure of the component the pixels (px) or percentage (%) units are used. If a measure is specified without unit identifier the value will be interpreted as a pixel value. The percentage value is interpreted as the percentage of the maximum value the component can fill up, which depends on where and how it is contained.

Example

125px, 50%, 400px

enumeration

An enumeration value is a set of predefined string values. They are often used when defining modes or types of which there is a limited number.

Example

Christmas, Easter, midsummer

color

A color value is a six-digit hexadecimal RGB [0 - 9 | a - f | A - F] value defining the red, green and blue color. Each color needs two digits ranging from 00 to FF defining the strength of the color. The color value can contain the optional number [#] prefix.

Example

#ff0000, #00ff00, #0000ff, #cccccc

reference

A reference value is an absolute or relative document path referencing to a document.

Example

http://www.xios3.com/file.xml, home://Document/file.xml, images/icons/home.png

css

A CSS value defines CSS (cascading style sheet) properties. The syntax of the CSS is a repeated sequence of property name, colon [:], property value and an ending semicolon [;].

Example

font-family:arial, verdana; font-size:11px; font-weight:bold;

Common attributes

The common attributes are component attributes that are managed by XIOS/3 directly and no the component code itself, and thus are avaialble on all components.

name

The most important common attribute is the name attribute, which is required on every component you want to interact with from the Process XML language. It is required when you bind data to the component, when you perform actions and mathods on it. You also need it to subscribe to events from the component.

Name	Type	Values	Description
name	string		The name of the component. It should be unique to the view.

width

The width attribute will set the component width in pixels or percentage value.

Name	Type	Values	Description
name	measure values	integer (px), percent (%)	The width of the component

height

The height attribute will set the component height in pixels or percentage value.

Name	Type	Values	Description
height	measure values	integer (px), percent (%)	The height of the component

title

The title attribute shows the specified text as a tool tip when the component is hovered.

Name	Type	Values	Description
title	string	Any string value	Tool-tip text.

label

The label attribute shows the specified text as a label above the component.

Name	Type	Values	Description
label	string	Any string value	The label text

lstyle

The lstyle attribute is used to style the label.

Name	Type	Values	Description
lstyle	string	CSS Syntax	Overrides the default label style

padding

The padding attribute defines the space between the component border and the component content.

Name	Type	Values	Description
padding	numerical	[0...n]	0

tabindex

The tabindex attribute defines in which order the tab button should focus on the components.

Name	Type	Values	Description
tabindex	numerical	[1...n]	0

style

The style attribute overrides CSS style information on the encompassing/major element of the component.

Name	Type	Values	Description
style	css	CSS Syntax	Overrides the appearance of the component using CSS

draggable

Declare this component as draggable. A draggable component can be dragged to a different component for drag and drop. A parent component will handle drag events for its children if they are not declared as draggable. The recipient component of the drop will get the current value of the component.

Name	Type	Values	Description
draggable	boolean	true, false	false

dragcomponent

Point out a component that should act as the dragged component if this component, or a child with no drag information, is targeted in a drag and drop event.

Name	Type	Values	Description
dragcomponent	string	component name	What component to use as drag component.

windowdrag

Indicate that dragging the component may be used to drag the window of the view around.

Name	Type	Values	Description
windowdrag	boolean	**true**, false	Can component be used to drag the window.

Events

A component end-user interaction triggers something called an event. A component event is a signal thrown to the application process XML trigger. The trigger will perform execution of the application logic for that specific signal, when the event occurs and the signal is thrown.

This section will present events in brief to give the reader an understanding of what the events should be used for. For a more detailed explanation of how to use events please read the documents Process Development and Process Development manual.

Event types

The event types that follow below are event listeners waiting for end-user interactions in the GUI to be performed, throwing events. They play a key role in how the components react and interact with the other components of an application.

AlignCenterRelease

This event occurs when text that is aligned in the center is released.

Applies To:

Editor, RichText

AlignCenterSelect

This event occurs when text that is aligned in the center is selected.

Applies To:

Editor, RichText

AlignJustifiedRelease

This event occurs when text that has justified alignment is released.

Applies To:

Editor, RichText

AlignJustifiedSelect

This event occurs when text that has justified alignment is selected.

Applies To:

Editor, RichText

AlignLeftRelease

This event occurs when text that is aligned to the left is released.

Applies To:

Editor, RichText

AlignLeftSelect

This event occurs when text that is aligned to the left is selected.

Applies To:

Editor, RichText

AlignRightRelease

This event occurs when text that is aligned to the right is released.

Applies To:

Editor, RichText

AlignRightSelect

This event occurs when text that is aligned to the right is selected.

Applies To:

Editor, RichText

Attached

This event occurs when a widget is attached to the widgetpane component.

Applies To:

WidgetPane

BGColorRelease

This event occurs when text that has background color is released.

Applies To:

Editor, RichText

BGColorSelect

This event occurs when text that has background color is selected.

Applies To:

Editor, RichText

Blur

This event occurs the user clicks outside the component after having set focus to it.

Applies To:

Input

BoldRelease

This event occurs when text that is bold is released.

Applies To:

Editor, RichText

BoldSelect

This event occurs when text that is bold is selected.

Applies To:

Editor, RichText

BulletListRelease

This event occurs when a bullet list has been released (unselected) inside a component that allows rich text editing.

Applies To:

Editor, RichText

BulletListSelect

This event occurs when a bullet list has been selected inside a component that allows rich text editing.

Applies To:

Editor, RichText

Change

This event occurs when the selected value of the component has been changed.

Applies To:

ComboBox, Input, RichText

Changed

This event occurs then the data bound to the component has been changed.

Applies To:

TextArea, XMLEditor

Click

This event occurs when the component has been clicked.

Applies To:

Map

Close

This event occurs when the component has been closed.

Applies To:

Console, Panel

Closing

In order for this event to occur the **handleClose** element has been added, and the value set to 'true', to the Close Operation when opening a view. The event occurs when the "X" is clicked to close the application. The application will not be clicked instead only the event will be fired allowing the developer to decide what to do.

Applies To:

View

Complete

This event occurs when the user trys to play a part of a media file but has not been loaded yet, which will make the component pause the media and throw an "Loading" event. It will wait until the the file loads enough to start playing from the requested part of the file and then continue playing, while throwing the "Complete" event.

Applies To:

Media

Completed

This event occurs after the content has been loaded

Applies To:

Browser

ContextMenu

This event occurs when the user right-clicks on the component.

Applies To:

ButtonBox, Calendar, Editor, Grid, Image, ListView, RichText, TabStrip, Textarea, Tree

DateChanged

This event occurs when a new date has been set.

Applies To:

Calendar, Date

DecorationRelease

This event occurs when text that is underlined is released.

Applies To:

Editor, RichText

DecorationSelect

This event occurs when text that is underlined is selected.

Applies To:

Editor, RichText

Delete

This event occurs when a selected item in the component has been deleted.

Applies To:

Grid, Tree

Detached

This event occurs when a widget has been detached from the widget pane.

Applies To:

WidgetPane

DoubleClick

This event occurs when the end-user double-clicks on the left mouse button.

Applies To:

Calendar, Editor, Grid, Image, Label, ListView, Render, RichText, Tree

drawingComplete

This event is occurs when the editor has completed the drawing of the XML data into visual objects.

Applies To:

Editor

DrawMode

This event is occurs when "Draw Mode" has been activated for the Editor component and is identified by the cursor becomming a crosshair.

Applies To:

Editor

drawModeOff

This event is occurs when "Draw Mode" has been deactivated for the Editor component and is identified by the cursor style returning to normal.

Applies To:

Editor

Drop

This event occurs when something is dropped onto the component.

Applies To:

Calendar, Console, Editor, Grid, Image, Label, List, ListView, Panel, RichText, Tree WidgetPane

Edit

This event occurs after editing has been performed on data in the component. To initiate editing see: callMethod edit.

Applies To:

ListView, Tree

EditEnd

This event occurs after the editfield is closed no matter if changes have been made.

Applies To:

ListView

End

This event occurs when the editor component is in fullscreen mode and the slide show has ended.

Applies To:

Editor

Empty

This event occurs when the component is emptied.

Applies To:

List

Escape

This event occurs when the ESCAPE button is hit on the keyboard while the component is selected.

Applies To:

Input

FGColorRelease

This event occurs when text that has foreground color has been released.

Applies To:

Editor, RichText

FGColorSelect

This event occurs when text that has foreground color has been selected.

Applies To:

Editor, RichText

Focus

This event occurs when focus has been set on the component.

Applies To:

ComboBox, Date, Input, RichText

FontFamilyRelease

This event occurs when text that has font-family style has been released.

Applies To:

Editor, RichText

FontFamilySelect

This event occurs when text that has font-family style has been selected.

Applies To:

Editor, RichText

FontSizeRelease

This event occurs when text that has font-size style has been released.

Applies To:

Editor, RichText

FontSizeSelect

This event occurs when text that has font-size style has been selected.

Applies To:

Editor, RichText

Fullscreen

This event occurs when the component is displayed in full screen.

Applies To:

Console

Header

This event occurs when the user clicks on the header of grid.

Applies To:

Grid

HeaderContextMenu

This event occurs when the user right-clicks on the header of grid.

Applies To:

Grid

Home

This event occurs when the editor component is in fullscreen mode and the slide show has been restarted.

Applies To:

Editor

Hover

This event occurs when the end-user hovers over the component with the pointer.

Applies To:

ButtonBox

HttpHyperlink

This event occurs when a hyperlink (http://) is clicked inside a richtext component.

Applies To:

RichText

Init

This event occurs when the component is rendered.

Applies To:

Accordion, Menu, Panel

ItalicRelease

This event occurs when text that is italic has been released.

Applies To:

Editor, RichText

ItalicSelect

This event occurs when text that is italic has been selected.

Applies To:

Editor, RichText

Key

This event occurs when a keyboard key has been pressed (onKeyDown) while the focus in on a supported component. The trigger does not contain any XML information.

Applies To:

Input, RichText, Textarea

Loading

This event occurs under different circumstances for different components.

The Browser component throws this event when the content of the component has started to load a webbpage.

The Media component throws this event when the user trys to play a part of a media file but has not been loaded yet, which will make the component pause the media.

Applies To:

Browser, Media

Ready

This event occurs when a component with external dependencies has been loaded and can accept interaction.

Applies To:

Map

Menu

This event is fired from two different components in two different ways.

For the calendar component the event occurs when the end-user right-clicks (requests a contextMenu) on the Calendar component, while not selecting any appointments (data).

For the panel component the event occurs when the end-user clicks on a small arrow, which is only visible when when de menu attribute is set to true.

Applies To:

Calendar, Menu, Panel

MediaEnded

This event is occurs when the media item has completed playback. The trigger (!) generated contains a reference to the folder containing the file.

Applies To:

Media

MediaRange

This event occurs when a song has started playback. The trigger (!) generated contains a reference to the folder containing the file.

Applies To:

Media

Move

This event occurs when the content of the component has been moved.

Applies To:

Map

MoveItem

This event occurs when the end-user drags a selected item.

Applies To:

List, ListView

Next

This event occurs when the editor component nagivates to the next slide.

Applies To:

Editor

NumberListSelect

This event occurs when a numbered list has been selected inside a RichTextEditor.

Applies To:

Editor, RichText

Paste

This event will occur when the user clicks on Ctrl+V. It will result in the paste-node is the trigger.

Applies To:

ListView

Previous

This event occurs when the editor component nagivates to the previous slide.

Applies To:

Editor

Release

This event occurs on mouse up or when something is deselected depending on the specific component.

Applies To:

ButtonBox, Calendar, List, ListView, Radio, RichText, TabStrip

Return

This event occurs when hitting the "Enter" key while the component is selected.

Applies To:

Calendar, Grid, Input, ListView, TextArea

Saved

This event occurs then the data bound to the component has been saved.

Applies To:

TextArea, XMLEditor

Select

This event occurs when the component has been selected. It is equivalent to onclick (and in some cases to onmousedown).

Applies To:

Accordion, Button, ButtonBox, Calendar, CheckBox, Clock, ComboBox, DataItem, Editor, Grid, Group, Image, Input, Item, Label, List, ListViev, Radio, Rating, Render, RichText, Slider, TabStrip, Tree

SelectedDate

This event occurs when a date is selected and displayed in the Calendar.

Applies To:

Calendar

SelectMode

This event occurs when "Select Mode" has been activated (and "Draw Mode" has been deactivated) for the Editor component and is identified by the cursor style returning to normal.

Applies To:

Editor

SelectedMore

This event occurs when an event has been extended from a single day to two or more while in single day view.

Applies To:

Calendar

SetActive

This event occurs once the view has been focused.

Applies To:

View

Show

This event occurs when the component is displayed.

Applies To:

Menu

SubjectMore

This event occurs when the "More Details" link is clicked in the Calendar Quick Event Scheduler.

Applies To:

Calendar

Tab

This event occurs when the tab button is hit.

Applies To:

Console

TriggerEvent

This event allows you to catch you own defined events from the ListView and Render components. In listview components the trigger will contain the selection data. This is however not available in the Render component.

Applies To:

ListView, Render

UnSelect

This event occurs when a shape has been unselected in the editor component.

Applies To:

Editor

UnSelection

This event occurs when the end-user unselects the component

Applies To:

XMLEditor

UpdateTime

This event occurs when a song has started to play and is fired continuously throughout the playback.

Applies To:

Media

Actions

A component state can be changed through something called an action. A component action is executed from any process XML logic by calling to the operation named action.

This section will present the concept of actions in brief to give the reader an understanding of what the actions should be used for. For a more detailed explanation of how to use and execute actions, please read the document Process Development and the Process Development manual.

Some actions are similar to methods. When that is the case understand that actions are handled faster than methods by XIOS/3 so if you can replace methods by actions you should.

Action types

The action types that follow are behaviors that change a state and sometimes the look and feel of a component. They play a key role in how the components behave and interact with the end-user.

activate

The activate action is used to set focus on a component. It works similarly to the focus method.

When a component is focused end-users normally can interact with it using the keybord.

Applies To:

RichText, TextArea, Tree

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="activate"/>
</operation>
```

Example: Process code for the add action

Related topics

focus

add

The add action is used to dynamically create (add) components inside other components. You can add Items to the Menu component or components to the Container component.

Applies To:

Container, Menu

Example

```

<operation name="action" value="#view1">
  <component name="component1" action="add" value="#view1#ListDynamicCreation"/>
</operation>
```

Example: Process code for the add action

Related topics

Container, Menu

clear

The clear action clears the value inside Input fields and Textarea. Useful for clearing form like application sections after data is saved.

Applies To:

Input, Textarea

Example

```
<operation name="action" value="#view">
  <component name="component1" action="clear"/>
  <component name="component2" action="clear"/>
</operation>
```

Example: Process code for the add action

Related topics

Input, Textarea

disable

The disable action will make a component grayed out in the GUI and disable all end-user interaction to the component.

This is for instance very useful when an application should disable the an Input component but should still be visualized in the GUI.

Applies To:

Button, ButtonBox, Calendar, Checkbox, Color, ComboBox, DataItem, Date, ExpanderItem, Grid, Group, Input, Item, Menu, Menu Item, RichText, Slider, Textarea

Example

```
<operation name="action" value="#view1">
  <component name="component2" action="disable"/>
</operation>
```

Example: Process code for the disable action

Related topics

enable

enable

The enable action is the opposite of the disable action and will return the component to its original state. The component will no longer be grayed out in the GUI and the end-user interaction will be set to normal.

Applies To:

Button, ButtonBox, Calendar, Checkbox, Color, ComboBox, DataItem, Date, ExpanderItem, Grid, Group, Input, Item, Menu, Menu Item, RichText, Slider, TextArea

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="enable"/>
</operation>
```

Example: Process code for the enable action

Related topics

disable

hide

The hide action will make a component disappear from the GUI until it is visualized again using the show action. This is useful when an application should both disable end-user interaction and to make the entire component hidden in the GUI.

Applies To:

Accordion, Browser, Button, ButtonBox, Calendar, Chart, CheckBox, Clock, Color, ComboBox, Console, Container, DataItem, Date, Expander, ExpanderItem, ExpanderMenu, Form, Grid, Group, Image, Info, Input, Label, List, ListView, Menu, Menu Item, MenuBar, Panel, Radio, Render, RichText, Slider, TabStrip, TextArea, XMLEditor, Toolbar, ExpanderMenu, Tree, Grid, Group

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="hide"/>
</operation>
```

Example: Process code for the hide action

Related topics

show

open

The open action is used to open ContextMenus.

Applies To:

Menu

Example

```
<operation name="action" value="#view1">
  <component name="menu1" action="open"/>
</operation>
```

Example: Process code for the open action

Related topics

Close

readonly

The readonly action sets the component in read only mode, meaning that it doesn't accept any user input that would modify the bound data. If the bound data is inherently read only the component will enter this mode on its own.

Related topics

readwrite

readwrite

The readwrite action sets the component in write enabled mode, meaning that it accepts user input that would modify the bound data.

Related topics

readonly

release

The release action is used when simulating an end-user interaction with the GUI, performing an event which deselects a component in the GUI without end-user participation. This is useful when an end-user interaction with a component should deselect a number of components simultaneously without requiring the end-user to deselect them all manually in the GUI.

This will trigger the Release event.

Applies To:

Button, ButtonBox

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="release"/>
</operation>
```

Example: Process code for the disable action

Related topics

select

select

The select action is used when simulating an end-user interaction with the GUI, performing an event which selects a component in the GUI without end-user participation. This is useful when an end-user interaction with a component should select a number of components simultaneously without requiring the end-user to select them all manually in the GUI.

Applies To:

Button, ButtonBox, Image, Menu Item

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="select"/>
</operation>
```

Example: Process code for the select action

Related topics

release

setValue

The setValue sets a simple string value to a component.

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="setValue">
    Composite of free text and {#view1#component2}.
  </component>
</operation>
```

Example: Process code for the setValue action

show

The show action is the opposite of the hide action and will return the component to its original state, making it visible and interactive for end-user events.

Applies To:

Accordion, Browser, Button, ButtonBox, Calendar, Chart, Checkbox, Clock, Color, Combobox, Console, Container, DataItem, Date, Expander, ExpanderItem, ExpanderMenu, Form, Grid, Group, Image, Info, Input, Label, List, ListView, Menu, MenuBar, Menu Item, Panel, Radio, Render, RichText, Slider, TabStrip, Textarea, Toolbar, Tree, XMLEditor,

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="show"/>
</operation>
```

Example: Process code for the show action

Related topics

hide

unbind

The unbind action is used to unbind data that has been bound to a component using the bind operation.

Applies To:

All components where data can be bound

Example

```
<operation name="action" value="#view1">
  <component name="component1" action="unbind"/>
</operation>
```

Example: Process code for the disable action

Related topics

bind

Methods

This is a reference section about all available component methods that can be used in Process XML language.

There are two ways of using methods, the callMethod operation or by appending .methodName() on an Expression.

Methods can be used to send information or to receive/extract information.

Using methods

As mentioned above there are two ways of using methods. Some methods are available via the callMethod operation, some are available using the .methodName() and some available via both.

Using callMethod

The callMethod operation allows you to send parameters to a component.

To se an example of such an operation, click here.

Appending .methodName()

When using Expressions you can also use methods.

It might looks like this #view#lvPlaylist.getParam('item'). This method, executed on a listview component called lvPlaylist.

Method types

Methods can be used to either sending information to a component or extracting information that the component might have.

In this reference section all available methods are listed, as well as on information on which components they apply.

activate

The activate-method is used to set focus to a component.

Applies To:

Input, ListView

```
<operation name="callMethod" value="#view1#component1">
  <method name="activate"/>
</operation>
```

Example: Process code for callMethod - activate

adjustVolume

The adjustVolume method is used to adjust the volume by a factor of 10. By sending in a positive integer, for example 1, the sound is increased by 10 of 100. If a negative integer is send the sound will decrease.

Applies To:

Media

Example:

```
<operation name="callMethod" value="#view1#media1">
  <method name="adjustVolume">
    <param type="integer">-1</param>
  </method>
</operation>
```

Example: Process code for callMethod - adjustVolume

appointmentColor

The appointmentColor method is used to set an color to an appointment in the Calendar component.

There is only a single parameter which defines the RBG color to set.

Applies To:

Calendar

```
<operation name="callMethod" value="#view1#calendar1">
  <method name="appointmentColor">
    <param type="string">ff0000</param>
  </method>
</operation>
```

attach

The attach-method is used to attach views to each other.

The first parameter defines the view to attach. Observe that the view still must be opened using the open operation.

The second parameter defines the position of the attached view. It can be set to "top", "bottom", "right" or "left".

The third parameter defines three offset parameters. The first value is the offset to the top, the second is the offset from the left and the third value is the stretch offset.

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="attach">
    <param type="string">#View1</param>
    <param type="string">bottom</param>
    <param type="string">50,0,0</param>
  </method>
</operation>
```

Example: Process code for callMethod - attach

callFlash

Makes a call to the flash container requesting the result of a specfic method, which is returned. The returned value is most likely a string but it depends on the flash file.

Applies To:

Flash

```
<alias name="flashValue" value="#view1#Flash1.callFlash("method24")"/>
```

Example: Process code for callFlash method

changeInterval

Changes the time interval that an hour should be divided to when displayed.

Applies To:

Calendar

```
<operation name="callMethod" value="#view1#calendar1">
  <method name="changeInterval">
    <param type="string">5</param>
  </method>
</operation>
```

Example: Process code for callMethod - changeInterval

clear

The clear-method is used to clear the contents of the component.

Applies To:

ComboBox, Textarea

```
<operation name="callMethod" value="#view1#widgetpane1">
  <method name="clear"/>
</operation>
```

Example: Process code for callMethod - clear

clearContent

The clearContent method is used to clear the contents of the container component.

Applies To:

Container

```
<operation name="callMethod" value="#view1#container1">
  <method name="clearContent"/>
</operation>
```

Example: Process code for callMethod - clear

clearSearchLines

The clearSearchLines method clears all search result lines in the XMLEditor.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="clearSearchLines"/>
</operation>
```

Example: Process code for callMethod - clearSearchLines

copyToClipboard

The copyToClipboard method allows for copying of XML data to the clipboard. It works similarly to the Ctrl+C (Copy) which is initiated by the end-user.

Applies To:

All components

```
<operation name="callMethod" value="#view">
  <method name="copyToClipboard">
    <param type="string">Component1</param>
  </method>
</operation>
```

Example: Process code for callMethod - copyToClipboard

deleteShape

Will delete to current selected shape in the Editor component.

Applies To:

Editor

```
<operation name="callMethod" value="#PresEd#slideEditor">
  <method name="deleteShape"/>
</operation>
```

Example: Process code for callMethod - deleteShape

deselect

The deselect method will deselect the selected row inside a component, such as the grid.

Applies To:

Grid

```
<operation name="callMethod" value="#view1#widgetpane1">
  <method name="deselect"/>
</operation>
```

Example: Process code for callMethod - deselect

detach

The detach method is used to detach views for each other.

The example below will detach view #two from view #one.

Applies To:

View

```
<operation name="callMethod" value="#one">
  <method name="detach">
    <param type="string">#two</param>
  </method>
</operation>
```

Example: Process code for callMethod - detach

disableAutoSpellcheck

The disableAutoSpellcheck method is used to disable automatic spellchecking in the RichText component.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richtext">
    <method name="disableAutoSpellcheck"/>
</operation>
```

Example: Process code for callMethod - enableAutoSpellcheck

edit

The edit-method is used to rename a document inside, for example, the listview component.

Applies To:

ListView, Tree

```
<operation name="callMethod" value="#view1#listView1">
  <method name="edit"/>
</operation>
```

Example: Process code for callMethod - edit

enableAutoSpellcheck

The enableAutoSpellcheck method is used to enable automatic spellchecking in the RichText component.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richtext">
    <method name="enableAutoSpellcheck"/>
</operation>
```

Example: Process code for callMethod - enableAutoSpellcheck

execCommand

The execCommand method is used to perform a command to a component.

Available commands: cut, paste, copy, undo, redo, selectAll

Applies To:

TextArea, XMLEditor

```
<operation name="callMethod" value="#view1#textarea1">
  <method name="execCommand">
    <param type="string">cut</param>
  </method>
</operation>
```

Example: Process code for callMethod - execCommand

executeCommand

The executeCommand method is used to pass a command to the RichText component.

Available commands are: InsertOrderedList, InsertUnorderedList, redo, selectAll, undo

Applies To:

RichText

```
<operation name="callMethod" value="#viewName#rte1">
  <method name="executeCommand">
    <param type="string">insertOrderdList</param>
  </method>
</operation>
```

Example: Process code for callMethod - executeCommand

executeScript

The executeScript method is used to execute a specific JavaScript function that has been included in to the view file.

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="executeScript">
    <param type="string">getValues</param>
    <param type="string">{$dataDoc#/settings/@versionId}</param>
  </method>
</operation>
```

Example: Process code for callMethod - executeScript

flashTab

The flashTab method is used to create that appearance that a tab is flashing by switching the selected and unselected visual mode of a tab.

The method has three parameters. The first parameter defines the name of the tab (panel component) that will start flashing and is required. Parameters two and three are optional, where two is the length of the flashing in milliseconds and the third is the speed of the flashing in milliseconds.

The default time length of the flashing is 2000 ms and the default flash delay is 200 ms.

Applies To:

TabStrip

```
<operation name="callMethod" value="#view1#tabstrip1">
  <method name="flashTab">
    <param type="string">examplepanel2</param>
    <param type="string">3000</param>
    <param type="string">100</param>
  </method>
</operation>
```

Example: Process code for callMethod - focus

focus

The focus-method is used to set focus on windows.

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="focus">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - focus

focusItem

The focusItem method focuses the selection on an item in the listview.

Applies To:

ListView

```
<operation name="callMethod" value="#view1#listview1">
  <method name="focusItem">
    <param type="string">{$dataDoc/settings/subscriptions/apprefs/appref[@id='']}</param>
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - focusItem

focusSelection

The focusSelection-method is used to set focus on single selection inside the component.

Applies To:

ListView

```
<operation name="callMethod" value="#view1#listview1">
  <method name="focusSelection">
    <param type="string">/settings/applicationlist/application[3]</param>
  </method>
</operation>
```

Example: Process code for callMethod - focus

getCurrentLabel

The getCurrentLabel method retrieves the label of the selected node in the tree component.

Applies To:

Tree

```
<alias name="labelOfSelection" value="{#view1#tree1.getCurrentLabel()}"/>
```

Example: Process code for the getCurrentLabel method

getCurrentPosition

The getCurrentPosition method retrieves the current time position of the playback.

Applies To:

Media

```
<alias name="playbackPosition" value="{#view1#media1.getCurrentPosition()}"/>
```

Example: Process code for the getCurrentPosition method

getDataLocation

The getDataLocation returns the data that is selected in the component.

Applies To:

Tree

```
<alias name="selection" value="#view1#tree1.getDataLocation()"/>
```

Example: Process code for the getCurrentLabel method

getDateValue

The getDateValue method returns the date that was clicked in the calendar (clock component)

Value returned is a string as such: 2019-03-05THH:MM:SS+0100

Applies To:

Clock

```
<alias name="selection" value="#view1#clock1.getDateValue()"/>
```

Example: Process code for the getDateValue method

getDimension

The getDimension method allows you to fetch the height and width of a window or component.

Applies To:

View

```
<alias name="myView" value="#view1"/>
<alias name="myViewHeight" value="$myView.getDimension().height"/>
<alias name="myViewWidth" value="$myView.getDimension().width"/>
<operation name="debug" value="$myViewHeight">alert</operation>
<operation name="debug" value="$myViewWidth">alert</operation>
```

Example: Process code to extract height and width of view window

Related:

setDimension

getEndDate

Retrieves the value of the dc:endDate from the selected appointment.

Applies To:

View

```
<alias name="appointmentEndDate" value="{#view#calendar.getEndDate()}"/>
```

Example: Process code for the getEndDate method

getEndTime

Retrieves the value of the dc:endTime from the selected appointment.

Applies To:

Calendar

```
<alias name="appointmentEndDate" value="{#view#calendar.getEndTime()}"/>
```

Example: Process code for the getEndTime method

getLat

Retrieves the latitude value from the selected coordinate.

Applies To:

Map

```
<alias name="markerLat" value="{#view#map.getLat()}"/>
```

Example: Process code for the getLat method

Related Topics:

getLng

getLineBottomMargin

The getLineBottomMargin method is used to read the paragraphs bottom margin.

Applies To:

RichText

```
<alias name="bottomMargin" value="{#view1#richtext1.getLineBottomMargin()}"/>
```

Example: Process code for getLineBottomMargin method

Related Topics:

getLineTopMargin

getLineTopMargin

The getLineTopMargin method is used to read the paragraphs top margin.

Applies To:

RichText

```
<alias name="topMargin" value="{#view1#richtext1.getLineTopMargin()}"/>
```

Example: Process code for getLineTopMargin method

Related Topics:

getLineBottomMargin

getLng

Retrieves the longitude value from the selected coordinate.

Applies To:

Map

```
<alias name="markerLng" value="{#view#map.getLng()}"/>
```

Example: Process code for the getLng method

Related Topics:

getLat

getMapType

Retrieves the map type currently used in the component.

Applies To:

Map

```
<alias name="mapType" value="{#view1#map.getMapType()}"/>
```

Example: Process code for the getMapType method

getZoom

Retrieves the zoom currently used in the component.

Applies To:

Map

```
<alias name="mapType" value="{#view1#map.getZoom()}"/>
```

Example: Process code for the getZoom method

getNodePath

The getNodePath method retrieves the full node path of the selection in the Tree component.

Applies To:

Tree

```
<alias name="fullPath" value="{#view1#tree1.getNodePath()}"/>
```

Example: Process code for the getNodePath method

getParam

The getParam method retrieves the value of the named parameter.

Applies To:

ListView

```
<alias name="lengthOfContent" value="#view1#listview1.getParam('keyword')"/>
```

Example: Process code for the getParam method

getPaste

The getPaste method is used by the Calendar component in order to retrieve clipboard information.

Applies To:

Calendar

```
<alias name="NewEndTime" value="{#view#calendar.getPaste('{$startDate} {$startTime}','{$endDate} {$endTime}','{$newStartDate} {$newStartTime}')}"/>
```

Example: Process code for the getPaste method

getSelected

Retrieves the XML data bound to the selected component rendered by a container.

Applies To:

Container

```
<alias name="content" value="#view1#container1.getSelected()"/>
```

Example: Process code for the getSelected method

getSelectedItem

Retrieves the selected item in a listview.

Applies To:

ListView

```
<alias name="content" value="#viewName#componentName.getSelectedItem()"/>
```

Example: Process code for the getSelectedItem method

getSelectedText

Retrieves the selected text in the component.

Applies To:

RichText

```
<alias name="selectedText" value="#view1#richtext1.getSelectedText()"/>
```

Example: Process code for getSelectedText method

getSelection

Retrieves the selection in the calendar component.

Applies To:

Calendar

```
<alias name="content" value="#view1#calendar1.getSelection()"/>
```

Example: Process code for the getSelection method

getSelectionLength

The getSelectionLength method retrieves the length of the content in the component.

Applies To:

Textarea, XMLEditor

```
<alias name="lengthOfContent" value="#view1#textarea1.getSelectionLength()"/>
```

Example: Process code for the getSelectionLength method

getSelectionValue

The getSelectionValue method retrieves the selected value of the content in the component.

Applies To:

Textarea, XMLEditor

```
<alias name="ValueOfTextarea" value="#view1#textarea1.getSelectionValue()"/>
```

Example: Process code for the getSelectionValue method

getSrcDataLocation

The getSrcDataLocation method retrives the document bound to the component, when regular Expressions might return xlinked documents.

Applies To:

Tree

```
<alias name="dataSrc" value="#view1#tree1.getSrcDataLocation()"/>"
```

Example: Process code for the getSrcDataLocation method

getStartDate

Retrieves the dc:startDate of the selection.

Applies To:

Calendar

```
<alias name="appointmentStartDate" value="{#view1#calendar1.getStartDate()}"/>
```

Example: Process code for the getStartDate method

getStartTime

Retrieves the dc:startTime of the selection.

Applies To:

Calendar

```
<alias name="appointmentStartTime" value="{#view1#calendar1.getStartTime()}"/>
```

Example: Process code for the getStartTime method

getText

Retrieves the content of the text() node.

Applies To:

Plate

```
<alias name="myTextValue" value="{#view1#plate2.getText()}"/>
```

Example: Process code for the getText method

getUpdateEnabled

The getUpdateEnabled method retrieves the state of the upateEnabled functionality. It returns a boolean value.

Applies To:

XMLEditor

```
<alias name="updateEnabled" value="{#view1#XMLEditor1.getUpdateEnabled()}"/>
```

Example: Process code for the getUpdateEnabled method

getValue

The getValue method retrieves the value of the content in the component.

Applies To:

Accordion, Browser, ButtonBox, CheckBox, Clock, Color, ComboBox, Date, Grid, Group, Image, Input, ListView, Menu Item Panel, Plate, Radio, Rating, RichText, Slider, TextArea

```
<alias name="ValueOfTextarea" value="#view1#textarea1.getValue()"/>
```

Example: Process code for the getValue method

getVolume

The getVolume method returns the volume of the media component as an integer.

Applies To:

Media

```
<alias name="mediaVolume" value="{#view1#mediaComponent1.getVolume()}"/>
```

Example: Process code for the getVolume method

gotoLine

The gotoLine method selects the defined line in the XMLEditor component.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="gotoLine">
    <param>12</param>
  </method>
</operation>
```

Example: Process code for the gotoLine method

hideColumn

This method is used to hide a visible column in a grid. The parameter sent contains the name of the column to hide.

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
  <method name="hideColumn">
    <param type="string">grid1Firstname</param>
  </method>
</operation>
```

Example: Process code for callMethod - hideColumn

haveSelection

The haveSelection method is used to check if a listview has a selection. The value returned is a string in the form of 'true' or 'false'.

Applies To:

ListView

```
<alias name="ListViewSelection" value="{#view1#listview1.haveSelection()}"/>
```

Example: Process code for method haveSelection

insertHyperlink

The insertHyperlink method is used to insert a hyperlink (home:// or http://) in to a richtext component.

The method takes two parameters, both are required. The first parameter determines the name of the link and the second parameter determines the actual URL.

Applies To:

RichText

```
<operation name="callMethod" value="#Write#richtextWrite">
  <method name="insertHyperlink">
    <param type="string">www.corren.se</param>
    <param type="string">http://www.corren.se</param>
  </method>
</operation>
```

Example: Process code for method haveSelection

insertImage

The insertImage is used to insert an insert an image in to a component. The image HTTP URL is provided as a parameter.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richtext1">
  <method name="insertImage">
    <param type="string">xios/icons/business/16x16/table.png</param>
  </method>
</operation>
```

Example: Process code for callMethod - insertImage

insertText

The insertText is used to insert an insert text in to a component. The text is provided as a parameter.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richtext1">
  <method name="insertText">
    <param type="string">Here is the text</param>
  </method>
</operation>
```

Example: Process code for callMethod - insertImage

lineNumbers

The lineNumbers method is used to either show or hide the line numbers in the XMLEditor.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="lineNumbers">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - focus

maximize

The maximize method is used to maximize windows.

Applies To:

View

```
<operation name="callMethod" value="#{!#@name}">
  <method name="maximize"/>
</operation>
```

Example: Process code for callMethod - maximize

minimize

The minimize method is used to minimize windows.

Applies To:

View

```
<operation name="callMethod" value="#{!#@name}">
  <method name="minimize"/>
</operation>
```

Example: Process code for callMethod - minimize

modStyle

The modStyle method is used to insert CSS styles in to components dealing with rich text.

The first parameter defines the name of the style parameter. Available values are "fontFamily", "fontWeight", "fontStyle", "fontSize", "textAlign"

The second parameter defined the value of the style parameter. This is different for every value in parameter one.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richText1">
  <method name="modStyle">
    <param type="string">fontWeight</param>
    <param type="string">normal</param>
  </method>
</operation>
```

Example: Process code for callMethod - modStyle

navigateUp

The navigateUp-method is used to navigate the parent of the current selection in the tree component.

Applies To:

Tree

```
<operation name="callMethod" value="#view1#Tree1">
  <method name="navigateUp"/>
</operation>
```

Example: Process code for callMethod - navigateUp

next

The next method is used start playing the next item in the playlist.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="next"/>
</operation>
```

Example: Process code for callMethod - next

nodeExpanded

The nodeExpanded method is used to determine if the selected node is expanded or not. It will return true or false.

Applies To:

Tree

```
<operation name="decision">
  <when test="'{#view1#Tree1.nodeExpanded()}'='true'" step="20"/>
  <otherwise step="21"/>
</operation>
```

Example: Process code for callMethod - nodeExpanded

pasteFromClipboard

The pasteFromClipboard method allows for paste of XML data from the clipboard. It works similarly to the Ctrl+V (Paste) which is initiated by the end-user.

Applies To:

All components

```
<operation name="callMethod" value="#view">
  <method name="pasteFromClipboard">
    <param type="string">Component1</param>
  </method>
</operation>
```

Example: Process code for callMethod - pasteFromClipboard

pause

The pause method is used to initiate a pause state for the media component.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="pause"/>
</operation>
```

Example: Process code for callMethod - pause

play

The play method is used to initiate a play state for the media component.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="play"/>
</operation>
```

Example: Process code for callMethod - play

previous

The previous method is used start playing the previous item in the playlist.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="previous"/>
</operation>
```

Example: Process code for callMethod - previous

print

The print method is used to print out a document.

Applies To:

Editor, RichText

```
<operation name="callMethod" value="#view1#editor1">
 <method name="print"/>
</operation>
```

Example: Process code for callMethod - print

printPresentation

The printPresentation method is used to print a Editor component document.

Applies To:

Editor

```
<operation name="callMethod" value="#view1#editor1">
  <method name="printPresentation" />
</operation>
```

Example: Process code for callMethod - print

release

The release-method is used to simulate the release command on components.

Applies To:

Button, ButtonBox

```
<operation name="callMethod" value="#view1#button1">
  <method name="release"/>
</operation>
```

Example: Process code for callMethod - release

releaseEnter

The releaseEnter-method simulates the release of the Enter-button.

Applies To:

Textarea, XMLEditor

```
<operation name="callMethod" value="#view1#component1">
  <method name="releaseEnter"/>
</operation>
```

Example: Process code for callMethod - releaseEnter

removeList

The removeList is used to remove a selected list in the RichText component.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#richtext">
  <method name="removeList"/>
</operation>
```

Example: Process code for callMethod - removeList

repeat

The repeat method is used to initiate a repeat state for the media component. The current playback will loop until aborted.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="repeat"/>
</operation>
```

Example: Process code for callMethod - repeat

resetSelection

The resetSelection method is used to reset the selecion in a component.

Applies To:

List, ListView

```
<operation name="callMethod" value="#viewName#componentName">
  <method name="resetSelection"/>
</operation>
```

Example: Process code for callMethod - release

restore

The restore method is used to restore a view (application) window to the desktop.

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="restore"/>
```

```
</operation>
```

Example: Process code for callMethod - release

restoreAppDlg

Restores the application window from the application toolbar to the desktop.

Applies To:

View

```
<operation name="callMethod" value="#{!#@name}">
  <method name="restoreAppDlg"/>
</operation>
```

Example: Process code for callMethod - restoreAppDlg

scale

Scales the VML/SVG rendering in the editor component.

Accepted parameter value is decimal value (using dot) such as "0.5" or "4.0". It can also be set to "auto" which will allow for autoscaling to fit editor component size.

Applies To:

Editor

```
<operation name="callMethod" value="#PresEd#slideEditor">
  <method name="scale">
    <param type="string">0.5</param>
  </method>
</operation>
```

Example: Process code for callMethod - scale

search

The search method allows you to search withing an XML document.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="search">
    <param type="string">XIOS</param>  <!--Match Whole Word-->
    <param type="Integer">0</param>       <!--Direction (0 or -1) -->
    <param type="Boolean">1</param>       <!--Match Whole Word-->
    <param type="Boolean">0</param>       <!--Match Case-->
  </method>
</operation>
```

Example: Process code for callMethod - search

searchMap

The searchMap method allows you to search for a location in a map.

Applies To:

Map

```
<operation name="callMethod" value="#view1#map">
  <method name="searchMap">
    <param type="string">{#view1#searchField}</param>
  </method>
</operation>
```

Example: Process code for callMethod - searchMap

selectAll

The selectAll method is used to select all rows.

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
  <method name="selectAll">
    <param type="string">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - selectAll

selectFirstRow

The selectFirstRow method is used to set selection to the first item in the grid

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
```

```
  <method name="selectFirstRow"/>
</operation>
```

Example: Process code for callMethod - selectFirstRow

selectLastRow

The selectLastRow method is used to set selection to the last item in the grid

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
  <method name="selectLastRow"/>
</operation>
```

Example: Process code for callMethod - selectLastRow

selectTab

The selectTab-method is used to set which tab is to be shown in the tabStrip from the process language.

Applies To:

TabStrip

```
<operation name="callMethod" value="#view1#tabStrip1">
  <method name="selectTab">
    <param type="string">tabClients</param>
  </method>
</operation>
```

Example: Process code for callMethod - selectTab

setBulletType

The setBulletType method is used to bullet types in components that utilize rich text.

The first parameter defines is the bullets should be removed. It can be set to "true" or "false". Default value is "false".

The second parameter decides the appearance of the bullets. It can be set to "disc", "square" or "circle". Default value is "circle".

The third parameter defines the position of the list. It can be set to "none", "inside" or "outside". The default value is "none".

The fourth parameter defines the color of the text and bullets. It can be set to any Hex-decimal value. The default color is black.

The fifth parameter is used to set a background image for the list by providing a URL to the image.

The sixth parameter is used to set the spacing between the list items.

Applies To:

Editor

```
<operation name="callMethod" value="#view1#richtext1">
  <method name="setBulletType">
    <param type="string">false</param>
    <param type="string">square</param>
    <param type="string"></param>
    <param type="string">#000000</param>
    <param type="string"></param>
    <param type="string"></param>
  </method>
</operation>
```

Example: Process code for callMethod - setBulletType

Removing lists:

```
<operation name="callMethod" value="#view1#richtext1">
  <method name="setBulletType">
    <param type="string">true</param>
    <param type="string">none</param>
  </method>
</operation>
```

Example: Process code for callMethod - setBulletType

setDimension

The setDimension method is used to change the dimension (width and height) of a view (application gui).

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="setDimension">
    <param type="integer">100</param>
    <param type="integer">200</param>
  </method>
</operation>
```

Example: Process code for callMethod - setDimension

Related:

getDimension

setEditableMode

The setEditableMode method is used to enable or disable editing of the component.

The parameter passed can be set to either true (editable) or false (not editable).

Applies To:

RichText, XMLEditor

```
<operation name="callMethod" value="#view1#xmleditor1">
  <method name="setEditableMode">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - setEditableMode

setFont

The setFont methods will set a new font on the component. The three parameters sent are Font-family, Font-Style and Font-Size.

Applies To:

Console, Textarea

```
<operation name="callMethod" value="#view1#Console1">
  <method name="setFont">
    <param type="string">Lucida Console</param>
    <param type="string">normal</param>
    <param type="string">12pt</param>
  </method>
</operation>
```

Example: Process code for callMethod - setFont

setIcon

The setIcon method allows the developer to set a new icon for a view window, by passing a parameter containing the URL to the new icon.

The method also allows a second parameter which regulates image scaling. The value passed can be either "scale" or left empty (default). This parameter can only be passed when using it with View's, not with Menu Item's.

Applies To:

ButtonBox, Menu Item, View

```
<operation name="callMethod" value="#view1">
  <method name="setIcon">
    <param type="string">xios/icons/applications/16x16/add.png</param>
    <param type="string">scale</param>
  </method>
</operation>
```

Example: Process code for callMethod - setIcon

setImage

The setImage-method allows the developer to set a background image on a view. This method is only applicable on view-files.

Applies To:

View

```
<operation name="callMethod" value="#view1">
  <method name="setImage">
    <param type="string">{#viewName#inputComponent}</param>
  </method>
</operation>
```

Example: Process code for callMethod - setImage

setLabel

The setLabel method allows the developer to change the label set for input fields.

Applies To:

Input

```
<operation name="callMethod" value="#view1#input2">
  <method name="setLabel">
    <param type="string">New Label</param>
  </method>
</operation>
```

Example: Process code for callMethod - setLabel

setLineMargin

The setLineMargin method allows the developer to change the line height in a richtext component.

The firstparameter defines the top margin and the second parameter defines the bottom margin.

Applies To:

RichText

```
<operation name="callMethod" value="#view1#component1">
  <method name="setLineMargin">
    <param type="string">0</param>
    <param type="string">0</param>
  </method>
</operation>
```

Example: Process code for callMethod - setLineMargin

setLineIndent

The setLineIndent method allows the developer to set an indentation in pixels in a richtext component.

The first parameter defined the left indentation and the second parameter defined the right indentation. If no text is selected then the indentation will be set for the entire component (all text).

Applies To:

RichText

```
<operation name="callMethod" value="#view1#component1">
  <method name="setLineIndent">
    <param type="string">10</param>
    <param type="string">0</param>
  </method>
</operation>
```

Example: Process code for callMethod - setLineIndent

setLook

The setLook method is used to change the appearance of the widgetpane component.

Applies To:

WidgetPane

```
<operation name="callMethod" value="#Desktop#deskWidgets">
  <method name="setLook">
    <param type="string">{#SettingsCopy#/settings/widgets/@sidebar}</param>
  </method>
</operation>
```

Example: Process code for callMethod - setLook

setMode

The setMode method is used to set the play mode of the media component. It can be set to either "audio" or "video".

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="setMode">
    <param type="string">audio</param>
  </method>
</operation>
```

Example: Process code for callMethod - setMode

setMute

The setMute method is used to mute the sound of the media component. It can be set to either true or false.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="setMute">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - setMute

setParam

This is used on the listview-component to send a parameter to the XSLT. In XSLT you use the lv:getParam-function to receive the parameter.

Applies To:

ListView, Render

```
<operation name="callMethod" value="#viewName#componentName">
  <method name="setParam">
    <param type="string">company</param>
    <param type="string">XIOS/3</param>
  </method>
</operation>
```

Example: Process code for callMethod - setParam

setRange

Sets the time range of the Calendar component.

Applies To:

Calendar

```
<operation name="callMethod" value="#view1#calendar1">
  <method name="setRange">
    <param type="string">07:00</param>
    <param type="string">19:00</param>
  </method>
</operation>
```

Example: Process XML code for callMethod - setRange

setSelect

The setSelect method is used to select a component, but has the option to decide if it should fire a Select event.

Applies To:

Menu Item

```
<operation name="callMethod" value="#view1#menuItem2">
  <method name="setSelect">
    <param type="string">false</param>
  </method>
</operation>
```

Example: Process XML code for callMethod - setSelect

setSelectedDate

The setSelectedDate method is used to selected a date in a calendar component, making it stand out from the non selected.

Applies To:

Calendar

```
<operation name="callMethod" value="#view1#calendar1">
  <method name="setSelectedDate">
    <param type="string">2019-03-05THH:MM:SS+0100</param>
  </method>
</operation>
```

Example: Process XML code for callMethod - setSelectedDate

setSelectionEx

The setSelectionEx method will set the selection expression in the tree component.

Applies To:

Tree

```
<operation name="callMethod" value="#view1#Tree1">
  <method name="setSelectionEx">
    <param type="string">/item[1]/p[2]</param>
  </method>
</operation>
```

Example: Process code for callMethod - setSelectionEx

setSelectionRoot

The setSelectionRoot method will set selection to the root.

Applies To:

Tree

```
<operation name="callMethod" value="#view1#Tree1">
  <method name="setSelectionRoot">
    <param type="string">/fs:folder[]</param>
  </method>
</operation>
```

Example: Process code for callMethod - setSelectionRoot

setSkin

The setSkin method is used to set XSL skin renderers for the clock component when in calendar mode.

Applies To:

Clock

```
<operation name="callMethod" value="#view#clCalendar">
  <method name="setSkin">
    <param type="string">apps/widgets/layout/cal_binder.xsl</param>
  </method>
</operation>
```

Example: Process code for callMethod - setSkin

setTitle

The setTitle method is used to set a title and/or subtitle to the view or the title of a panel.

The subtitle can only be set for views. Additionally you can only only send a second parameter which will just update the subtitle. If both a title and subtitle are used the "-" delimiter will be added automatically.

Applies To:

Panel, View

```
<operation name="callMethod" value="#view1">
  <method name="setTitle">
    <param type="string">Application Name</param>
    <param type="string">Subtitle Text</param>
  </method>
</operation>
```

Example: Process code for callMethod - setTitle

setType

Sets the display type of the calendar component.

Applies To:

Calendar

```
<operation name="callMethod" value="#Calendar#Calendar">
  <method name="setType">
    <param type="string">7week</param>
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - setType

setUpdateEnabled

The setUpdateEnabled method set the value of the UpdateEnabled functionality. The parameter is either true or false.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="setUpdateEnabled">
    <param>false</param>
  </method>
</operation>
```

Example: Process code for callMethod - setParam

setValue

This method will set a value to a component.

Applies To:

Accordion, Browser, Button, Color, ComboBox, Editor, Flash, Group, Image, Input, Label, List, Media, Menu, Radio, Rating, Slider

```
<operation name="callMethod" value="#viewName#componentName">
  <method name="setValue">
    <param type="string">tasks</param>
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - setValue

When using the setValue method for the media component you can set the position of the playback in seconds.

```
<operation name="callMethod" value="#view1#mediaComponent">
  <method name="setValue">
    <param type="integer">90</param>
  </method>
</operation>
```

Example: Process code for callMethod - setValue - Will set position to 1:30 minutes into the playback

While changing the value of the SWF-file to play in the Flash component then the first (and only) parameter should contain the http-url to the file.

```
<operation name="callMethod" value="#view1#flash1">
  <method name="setValue">
    <param type="string">http://www.example.com/file.swf</param>
  </method>
</operation>
```

Example: Example showing how to change the swf-file displayed in the Flash component

setView

The setView-method is used to change XSLT files in render components while keeping the data bound to the component.

The change is made by replacing the name of the file (excluding the file type suffix), furthermore they should be contained within the same folder.

Applies To:

ListView, Render

```
<operation name="callMethod" value="#view1#listview1">
```

```
  <method name="setView">
    <param type="string">documents_list</param>
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - setView

setVolume

The setVolume method is used to set the sound of the media component. It can be set to a value between 0-100.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="setVolume">
    <param type="integer">80</param>
  </method>
</operation>
```

Example: Process code for callMethod - setMute

setWeekStartDay

Sets the first day of the week for the calendar component. The value can be either Monday or Sunday, where Monday is default.

Applies To:

Calendar

```
<operation name="callMethod" value="#view1#calendar1">
  <method name="setWeekStartDay">
    <param type="string">Monday</param>
  </method>
</operation>
```

Example: Process code for callMethod - setWeekStartDay

showColumn

This method is used to show a hidden column in a grid. The parameter sent contains the name of the column to display.

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
  <method name="showColumn">
    <param type="string">grid1Firstname</param>
  </method>
</operation>
```

Example: Process code for callMethod - showColumn

showControls

The showControls method is used to define if controls should be visible for the component.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="showControls">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - showControls

shuffle

The shuffle method is used to initiate a shuffle state for the media component. Plays items in playlist in a random mode.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="shuffle">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - shuffle

sortColumn

This method is used to define how a grid should be sorted. The first parameter is the column name and the second parameter is the type or sorting to perform; ascending or descending.

Applies To:

Grid

```
<operation name="callMethod" value="#view1#grid1">
  <method name="sortColumn">
    <param type="string">grid1Firstname</param>
    <param type="string">ascending</param>
  </method>
</operation>
```

Example: Process code for callMethod - sortColumn

stop

The stop method is used to initiate a stop state for the media component.

Applies To:

Media

```
<operation name="callMethod" value="#view1#mediaComponent1">
  <method name="stop"/>
</operation>
```

Example: Process code for callMethod - stop

takeEnter

The takeEnter-method simulates the press of the Enter button. This makes it possible to have focus on several components.

Applies To:

Textarea, XMLEditor

```
<operation name="callMethod" value="$appComp">
  <method name="takeEnter">
    <param type="string">#FR#findNext</param>
  </method>
</operation>
```

Example: Process code for callMethod - update

toggleCheckable

Toggles the mode of the menu Item to true or false. It simulates a selection made on the component.

Applies To:

Menu Item

```
<operation name="callMethod" value="#view1#mItemComments">
  <method name="toggleCheckable">
    <param type="boolean">true</param>
  </method>
</operation>
```

Example: Process code for callMethod - toggleCheckable

toggleNode

The toggleNode-method toggles a node in the tree component.

Applies To:

Tree

```
<operation name="callMethod" value="#view1#Tree1">
  <method name="toggleNode"/>
</operation>
```

Example: Process code for callMethod - update

update

The update-method updates the data bound to a component.

In the case of the Panel component it can be used to update the title of the panel (if such exists).

Applies To:

Grid, ListView, Panel

```
<operation name="callMethod" value="#viewName#Component">
  <method name="update"/>
</operation>
```

Example: Process code for callMethod - update

updateData

Makes an update to the data document bound to the component.

Applies To:

XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="updateData">
    <param>true</param>
  </method>
</operation>
```

Example: Process code for callMethod - updateData

updateRedraw

The updateRedraw-method redraws the component thus applying new data (or parameters) that might have been passed to it.

Applies To:

ListView, Render

```
<operation name="callMethod" value="#viewName#componentName">
  <method name="updateRedraw"/>
</operation>
```

Example: Process code for callMethod - updateRedraw

wordWrap

The wordWrap-method methods defines if what type of word-wrap should be used. There are three options; hard, soft or off.

Applies To:

RichText, XMLEditor

```
<operation name="callMethod" value="#view1#XMLEditor1">
  <method name="wordWrap">
    <param type="string">hard</param>
  </method>
</operation>
```

Example: Process code for callMethod - wordWrap

Rules

The rule element is used to define how a data driven component should interpret the bound data. Every component uses their own rules but have rule element and match attribute in common with each other. The matching is done by using Expressions to match bound data

Observe that some components actually require rule elements in order to display the data, such as the Tree component.

Since rules are handled by the components themselves the data is left unaltered thus several component can show the same data is several different ways and changes in one are propagated to all the components and displayed accordingly.

Note: If you are trying to match elements or attributes which use namespaces then you might need to declare that namespace on the element. If you do not then the rule matching might not fail, and no error messages might be displayed.

Applies To:

Clock, ComboBox, Container, DataItem, Form, Grid, Info, Tree

Example:

```
<combobox name="CB" width="150" height="20" editable="false">
  <plate name="CBplate" height="30">
    <rule match="component">
      <item>{@name}</item>
    </rule>
  </plate>
</combobox>
```

Example: Rule in Combobox component

```
<components>
  <component name="Accordion" tag="Component" group="UI XML">
    <documentation>
      <description>Accordion description.</description>
    </documentation>
  </component>
  <component name="Browser" tag="Component" group="UI XML">
    <documentation>
      <description>Browser description.</description>
    </documentation>
  </component>
  <component name="Button" tag="Component" group="UI XML">
    <documentation>
      <description>Button description.</description>
    </documentation>
  </component>
</components>
```

Example: Data XML for ComboBox Example

Look and Feel

In UI XML you as a developer is able to make use of CSS (Cascading Style Sheets) to override the current style that apply to components using the style attribute which is available on all components. CSS is written by using the notation key:value; repeatedly.

In XSLT driven components, such as the ListView or Render you are able to use CSS classes. These can be created inside an external CSS file which is included in your applications view file, using the style element. There is also support for inline CSS which means that you can write your CSS directly in the view file. If you have a small application or a widget with some CSS it is recommended that you use inline CSS since it reduces the number of requests made to the server, making your application load faster. Have this in mind when creating widgets since they often are loaded on system boot up.

If a href attribute is not supplied the inline CSS will be used. If however a href attribute is supplied there inline CSS will be ignored and the external CSS will be used instead.

For those that are not familiar with CSS classes they are applied on HTML elements (in XSLT files) using a class attribute.

Attributes

Optional attributes

Name	Type	Values	Description
href	string	URL	The path to the external CSS file to use.

Example 1: External CSS

```
<view name="view1" title="Hello World" version="1.0" height="400" width="400" icon="icon://earth">
  <style href="home://Documents/helloWorld.css"/>

  <panel height="auto" width="auto">
    <render height="auto" width="auto" href="home://Documents/render.xsl"/>
  </panel>
</view>/>
```

Example: This view file is importing the CSS file making it available to the render component where the classes are used

```
.bold {
  font-weight: bold;
}
.headlineBlue {
  font-weight: bold;
  font-size: 16px;
  color: blue;
}
```

Example: This is the CSS file that we will import

Example 2: Inline CSS

```
<view name="view1" title="Hello World" version="1.0" height="400" width="400" icon="xios/icons/network/16x16/earth.png">
    <style>
    .bold {
      font-weight: bold;
    }

    .headlineBlue {
      font-weight: bold;
      font-size: 16px;
      color: blue;
    }
    </style>

    <panel height="auto" width="auto">
      <render height="auto" width="auto" href="home://Documents/render.xsl"/>
    </panel>
</view>/>
```

Example: Using inline CSS

Process Development

In this section we will cover everything about Process Development and is divided in to sections of relevance.

Process XML is the logic part of XIOS/3 Application Development. When considering the MVC design pattern, Process XML equals to the Control.

What this section covers

This part of the documentation will cover all parts of Process XML. It starts with a section about aliases, which are XIOS/3 equivalents of variables. We cover how they are constructed, how to use them and what system aliases are available.

The Event Model, which is the foundation of end-user interactivity with components as well as with the XIOS/3 system. The section provides you with examples of binding components to each other as well as how to work with triggers and selections. In this section is also an event section which provides information about View and System events.

There is also a section explaining what Steps are, how to use them and how they should be interpreted. Since all operations are children of steps this is an important section to understand.

When trying to understand application logic, our section on Expressions is very important. It contains key information about how components can interact and also includes examples.

Overall Structure

The sections covers the corner stones and the construction of the Process XML language. First the root element, process, and how to use it. We then explain how trigger elements work, what their purpose is and how events are captured.

A detailed section about the step element where and a short primer on the operation element is also provided.

An extensive section about Aliases explaining what they are, how they are used as well as what other purposes they can be used for. The Alias section contains two chapters; Alias Types and System Aliases.

Process

The process element is the root element for all process files (Process XML files).

Usage

This element has two required attributes, the name attribute which must be unique within the application package and the version attribute which refers to which Process XML language version is used. The reason why the name attribute must be unique is because it is used as a identifier and that two files with the same name can't be active at the same time. Like with the view element there are author and description attributes, in this case used only for internal purposes.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	Any string value	The name of your process file
version	enumeration	1.0	File version (personal use)

Optional attributes

Name	Type	Values	Description
author	string	Any string value	Author of the file
description	string	Any string value	File description

Triggers

The event listeners are used to set up a event catcher; so when interaction is made in the view file (UI XML) these listeners will be activated and then execute a specified step and all operations inside that step.

These elements are child elements of the process element.

Events can be made by the user in the form of component events, which are made when the user interacts with components. Users can also induce so called application events by using keyboard shortcuts while having he application selected.

There are also system events which are made by the XIOS/3 system and are not application specific.

Attributes

Required attributes

Name	Type	Values	Description
event	enumeration	predefined event label	Name of the event to listen for from the lists of
step	string	Any string value	Step id to execute

Optional attributes

Name	Type	Values	Description
component	string	Any string value	Name of component making event
communicator	string	Any string value	Name of communicator
view	string	Any string value	Name of view file where the event is originating from

Available Events

Component Events, System Events, View Events

Syntax and Examples

Syntax

```
<trigger view="[string]" component="[string]" event="[enumeration]" step="[string]"/>
```

Example

```
<?xml version="1.0"?>
<process name="calculator" description="Main process file for calculator">
  <trigger view="calculator" component="btn_Sum" event="Select" step="Summerize"/>
  <trigger view="calculator" component="btn_Add" event="Select" step="Addition"/>
  <trigger view="calculator" event="Ctrl+A" step="Addition"/>

  <step id="Addition" name="Perform addition">
    ...
  </step>
</process>
```

Related Topics

Steps

Steps are an important part of XIOS/3 Application Development. They act as logic clusters since they can contain several operations. There are some simple rules and guidelines when it comes to steps. They are do not forget to have a step id called 1 and do not forget that a step id must be unique in a Process XML file.

You can easily navigate from one step to another by using operations such as call or decision.

When opening steps you can also provide the operation number to which you would like to start execution from. Se example at the bottom.

Attributes

Required Attributes

Name	Type	Values	Description
id	string	Any string value	A step unique identifier
name	string	Any string value	An optional name which can act as a description for the step

Optional Attributes

Name	Type	Values	Description
error	string	Any string value	Id of step to call when error occurs while processing step

success	string	Any string value	Id of step to call when step is complated successfully

Syntax

```
<step id="identifier" name="string">
  <operation name="enumeration" value="expression|string"/>
</step>
```

Example

```
 1 <?xml version="1.0"?>
 2 <process name="applicationName" description="Logic for applicationName" author="XIOS/3">
 3   <step id="1" name="Initiate Application">
 4     <operation name="open" value="home://Examples/view1.xml"/>
 5     <operation name="call" value="Config"/>
 6     <operation name="call" value="2:2"/>
 7   </step>
 8
 9   <step id="2" name="Alert Values">
10     <operation name="debug" value="$pi">alert:plain</operation>
11     <operation name="debug" value="$mediaplayerPath">alert:plain</operation>
12   </step>
13
14   <step id="Config" name="Configure Application" sucess="2" error="2">
15     <alias name="pi" value="3.14" model="global"/>
16     <alias name="mediaplayerPath" value="apps/mediaplayer/mediaplayer.xml" model="global"/>
17   </step>
18 </process>
```

Line 01-02

The first two lines contain the XML Processing Instruction on the first line and then the process element on the second row. The process element is the root element of every Process XML file.

Line 03-07

Our first step element must have the id value 1. The reason why we will set it to 1 is because it is the default. When opening process files and not supplying an id to start it will assume that you want to start with step 1. We have named our step appropriately "Initiate Application".

Our first operation is open, which will open our view file located at a fictitious path. After it has been opened, we move to our next operation which is a call operation. This means that it will start another step, in this case a step called Config, which starts at line 13. The third operation in step 1 will call on step 2 and start executing all operations starting with the second operation.

So the logic is that is should open a file called view1.xml, then call on a step called Config and then finally call step 2 and start executing from operation 2.

Line 09-12

This step is called 2 and it will, as its name attribute implies, alert our values. When we called this step we added information that we want to execute all operations starting from the second, using the following information; (2:2). This step will thus only alert (JavaScript style) our second alias value using the debug operation. The first will be ignored.

As you can see we use the variables $pi and $mediaplayerPath, which are not set until the Config step. However since we call the Config step before this (2) they are available. If they were not they would have no value.

Line 13-16

Our last step is the Config step; which as explained earlier is called before step 2 since we need the aliases that are set in the Config step. Here we simple use alias elements and set their name and value. The first we will name pi and set its value to 3.14, and then the mediaplayerPath alias which will be set as an URL. Both of the aliases have an attribute called model; this is set to global since we want to ensure that these are available in every step in the process file.

Aliases

Aliases are powerful and flexible equivalents to variables and are used in the Process XML language. It is a reference to a value or an XML node set.

Usage

With aliases you can store strings, integers, paths and XML data. They are global in scope by default, which means that they are available to all steps in the process XML file. There is also support for local scope, which makes the information stored in the alias only available in a single step.

Alias are also used to open files from the file system and can be used to create new XML node sets.

An alias is an Event Object that is either a document or a string value. If it is a document then if would consists of three parts; data, xpath and selection. (see Event Object in Event Model section).

The alias element has three attributes. Here the name attribute defines the name of the attribute, which is used when calling on the alias. An alias is available using a dollarsign followed by the value of the name attribute. Keep in mind that aliases can be overwritten by defining them again using another value.

When using the alias to store information something must be provided in the value attribute or in the case that the model attribute is set to define, in the value child element. Besides supporting integers and strings alias can also support XML data which is provided using Expressions.

Model defines how the system should interpret your alias. If it is a value (integer or string) or if it contains XML data. Here you should note that is the alias contains XML data and the model is set to value that it will convert the data in to a string and omit all elements, leaving only the text() nodes. As we mentioned earlier alias can use global or local scope. This might induce the need to make local scope aliases in to global, which is done by setting the model to global.

There are several model options that are not used to when storing information, but instead to make the system execute an action. Here the model="new" is used to create a new XML node set that can be used temporarily until it is saved. Using the model="open" will display an "Open file" modal dialogue so that the end-user can select a file. Parameters can also be provided when opening files, which enables the developer to rename the buttons visible to the end-user and also what mime-type that can be selected. The model="folder" works like the model="open" but will instead allow the end-user to select a folder. Note that when opening an XML document the entire document will be available, but when opening something else (i.e. image file) only meta data will be available. When opening a folder you will only gain a string (the path) to the folder selected in the folder selector.

Using the model="define" you can define the base and selection of the data.

When a document is referenced to in XIOS/3 the system will begin to transaction handle the file. If you want this feature to be turned off then set the model attribute to "local". Observe that this must be declared the first the document is included. An included document cannot be "converted" to a local document. Advantages that model local offers are that less server load and bandwidth are used.

Attributes

Name	Type	Values	Description
name	string	Any string value	The name of the alias
value	string	string, integer, Expression	Decides what the alias will contain
model	enumeration	**data**, global, value, newsilent, new, open, folder, define, browse, local	How the alias should be interpreted by the system

Syntax and Examples

```
<alias name="[string]" value="[string|integer|expression]" model="[enumeration]/>
```

Example: Syntax for a common alias.

```
<alias name="myXMLalias" value="{xml(#view1#component1)}"/>
```

Example: This will evalute the expression but will keep it as XML data.

```
<alias name="dataDoc" model="open">
  <dialog title="Open" folder="home://Documents">
    <label name="ok">Open File</label>
    <label name="cancel">Cancel</label>
    <type mime="text/xml"/>
    <type mime="text/plain"/>
  </dialog>
</alias>

```

Example: Opening a file.

```
<alias name="dataDoc" model="folder">
  <dialog title="Open" folder="home://Documents">
    <label name="ok">Select Folder</label>
    <label name="cancel">Cancel</label>
    <label name="createFolder">Create Folder</label>
    <type mime="text/xml"/>
  </dialog>
</alias>

```

Example: Selecting a folder.

```
<alias name="tmpSelection" model="define">
  <document>#tmp1</document>
  <base>/components[1]</base>
  <selection>
    <item select="/components[1]/component[2]"/>
    <item select="/components[1]/component[3]"/>
    <item select="/components[1]/component[5]"/>
  </selection>
</alias>
```

Example: Alias using model define.

```
<alias name="targetFile" model="browse">
  <dialog title="Select Shortcut Target">
    <label name="ok">Select</label>
    <label name="cancel">Cancel</label>
    <type mime="text/xml"/>
  </dialog>
</alias>
```

Example: Alias model browse allows you to select any file or folder in the filesystem which then becomes available in the defined alias.

Alias Types

Common Use

The code below will hold XML data that is temporarily stored in the trigger. The reason that you might want to store your trigger-value is because the trigger constantly changes. A global variables will be created and the value will be available in all steps.

```
<alias name="selected" value="!" model="data"/>
<operation name="debug" value="$selected">alert</operation>
```

Example: Getting the selection data in the trigger

Examples

data

"data" will set the data document and selection (if any exists) of the reference document. It is the default model used.

```
<alias name="LvSelection" value="!" model="data"/>
```

Example: Alias with mode set to data

global

"global" will make the alias available globally. The indented use for this is when dealing with local aliases set by parameters using the call operation.

```
<alias name="DataDoc" value="!" model="global"/>
```

Example: Alias with model set to global

local

"local" is used to create non-transactioned temporary documents. This means that the XIOS/3 TransactionManager will not be used.

new

"new" is used to create a new temporary data document on the fly by adding the data document as child elements of the alias element. It can be saved.

```

        <alias name="DataDoc" model="new">
  <clients/>
</alias>
```

Example: Alias with model set to new

newsilent

"newsilent" is used to create a new temporary data document, without notifying the user about the procedure.

```

        <alias name="SaveData" model="newsilent">
  <mydata/>
</alias>
```

Example: Alias with model set to newsilent

string

"string" will allow the alias text to be handled as a string. Thus a number of instructions can be performed on the string value such as regular expressions and replacing of substrings.

In the above example the alias $URLstring will contain a string that looks like a URL. Our second alias $URL will contain a data document of the file.

value

"value" will convert the reference to a string (omitting the XML).

```
<alias name="CompanyName" value="{#view1#component1}" model="value"/>
```

Example: Alias with model set to value

System Aliases

In XIOS/3 there are some alises that are implicit (predefined) and can be used directly in the Process XML language without the need to define a value to them. Some retrieve information directly from the user computer, such as time and date. These are reserved and cannot be overwritten.

date

The date alias will return the current date of the user running XIOS/3.

```
<operation name="debug" value="$date">alert</operation>
```

Example: 2007-12-13

time

The time alias will return the local time of the user running XIOS/3.

```
<operation name="debug" value="$time">alert</operation>
```

Example: 11:19

milliseconds

The tm alias will return the number of milliseconds from January 1, 1970 GMT as an integer.

This alias is very useful when you must create a unique name.

To make $tm work first place it inside a variable. Then when using the new alias always place it inside curley brackets.

```

<alias name="ctm" value="$tm"/>
<operation name="debug" value="{$ctm}">alert</operation>
```

Example: 1197541185125

widget

This alias contains the widget settings of the widget using it. The result will be displayed in XML data where the root is atom:entry. Note that if the widget is not defined in the settings.xml that this system alias will not contain any information.

```
<operation name="debug" value="$widget">alert</operation>
```

Example: [XML DATA]

instance

The instance alias will return the instance of the application package opened.

```
<operation name="debug" value="$instance">alert</operation>
```

Example: 2

Note that an application can limit the number of instances that can be active and that a specific application instance can be reached using "#appname_{$instance}"

package

The package alias will contain the XML of the application file. Observe however that when you are using this alias that you must use relative path and never absolute path.

Using this alias you can open views and process files by adding an expression after the variable.

```
<operation name="debug" value="$package">alert</operation>
```

Example: [XML DATA]

home

The home alias is available when using application packages and when the home element has been defined in the settings element of the application file.

```
<operation name="debug" value="$home">alert</operation>
```

Example [XML DATA in the form of a filelisting of the folder]

Operation

The operation element is used to perform application logic on one or several UI components.

Usage

Operation is a child element of the step element and will therefore be executed once that step is executed by the system.

In the section "Using Operations" we explain the attributes of operations as well as a complete operation reference. The type of operation (name) and on what component or view it should be executed (value) are provided as attributes. For things to make sense we need to separate the meaning of operations and the operation element in this documentation. While there is only one operation element there are several different types of operations (such as call, bind or debug). You define the option to use in the name attribute of the operation element.

The value attribute which can be found on most operations will always evaluate the value placed in it, whether it is an alias, an expression or a string.

Event Model

The Event Model is central to understanding how XIOS/3 programming works. It is the foundation for end-user interaction with XIOS/3 applications.

Events are emitted when the end-user interacts with XIOS/3 and your application. If data is bound to the object from which the event was emitted it too can be used. The Event Object can also be emitted using Expressions. In this case no event type is needed and like with end-user created events then state of the component is provided.

Event Model Flow Chart

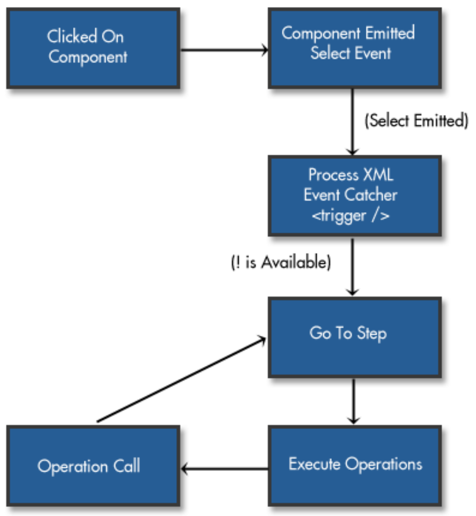


Figure: The Event Model Flow Chart. Interaction with components will trigger events which in turn are used to execute application logic in the form of operations.

Event Object

The Event Object is emitted every time a component is clicked; the developer requests it from a component or when a user perform a shortcut command.

Every Event Object is made up of five parts. One of these is the event type, which contains information about what event triggered the Object. The second part is the **data document** (abbreviated dataDoc) which contains a URL to the XML data document that is bound to the component that made the event. Note that when events are made by components that are not bound to data, system events or view events that there is no dataDoc, and that therefore can't be any Base or Selection either.

In the Xpath we find the basepath to which the component is bound to. The selection part contains a xpath of the selection in the data starting from the basepath. The fifth part is the value. If a text()-node is selected a text will appear here, if not it will be empty.

Name	Type	Value	Description
Type	enumeration	Component Events, System Events, View Events	The type of event made
dataDoc	string	path	An URL to the XML data document
Xpath	string	match	The basepath of the bound data
Selection	string	match	The xpath selection in the XML document, starting from the basepath
Value	string	Any string value	The text node of the selection (if it exists)

Table: The Contents of the Event Object

Event Types

There are three different types of events; Component Events, System Events and View Events. They differentiate by what created them. In the end they are all made by user interaction, however they are made by different parts of XIOS/3 and thus made.

Component Events are made from when the user interacts with a component, like a button. It will (when selected) create an Event Object with the type Select.

System Events are made by the XIOS/3 system. These include Login and Authorize events.

View Events are specific to views and are basically shortcut events such as CTRL+S or ESCAPE. These can also be captured by event listeners.

Event Capturing

See trigger element section.

Component Events

Binding components to other components

When binding one component to another you create an event object, that does not contain any event type. The real advantage of binding data like this is that changes made to the XML file will be updated directly in both components when making a change to the data with either component.

```

        <?xml version="1.0"?>
<process name="musicplayer" description="musicplayer application" author="XIOS/3">

  <step id="100" name="Bind data to combobox using grid">
    <operation name="bind" value="#musicplayer#gridMusic">
      <component view="musicplayer" name="cb_artists" select="/"/>
    </operation>
  </step>
</process>
```

Working with the !

In the example below we first bind the stations.xml document to a grid component called gridMusic. We then set up an event listener that will listen for the Select event made in the grid component. This is made when the user selects a value in the grid.

The first step will populate the grid component with radio stations (using thebind operation). We bind the data from the root; which is radio.

The second step will be executed by the event listener. When a selection is made in the grid a single station node will be selected. In step 101 we bind the event object selection to our other two components. The description attribute is bound to the input field and the entire station element is bound to the listview.

The XML data

```

        <?xml version="1.0"?>
<radio>
  <station name="Example 1" url="http://example.com/" description="Example music for all ages."/>
  <station name="Example 2" url="http://example2.com/music/" description="Jazz Music by Example Ltd."/>
</radio>
```

The Process XML

```

        <?xml version="1.0"?>
<process name="musicplayer" description="musicplayer application" author="XIOS/3">
  <trigger view="musicplayer" component="gridMusic" event="Select" step="101"/>

  <step id="1" name="Bind data to grid component">
    <operation name="bind" value="home://data/stations.xml">
      <component view="musicplayer" name="gridMusic" select="/radio"/>
    </operation>
  </step>

  <step id="101" name="Bind data to input field">
    <operation name="bind" value="!">
      <component view="musicplayer" name="inputEdit" select="/station/@description"/>
      <component viwe="musicplayer" name="listviewPlayer" select="/station"/>
  </operation>
 </step>
</process>
```

View Events

There are events that are specific to views and not components. Currently the majority of events available are shortcut events that allows the application developers to make shortcut commands in their application. There are however some commands that are reserved by XIOS/3. A list of these is posted below.

The shortcut events are produced when a user interacts with an application using shortcut keys such as Ctrl+S for save and Ctrl+C for copy. XIOS/3 allows for capturing these events and then trigger operations.

XIOS/3 can capture a combination of control, shift, alt and meta key. Just combine them as "Ctrl+", "Shift+", "Alt+" and "Meta+" prefix to unmodified key label, e.g. "Ctrl+Alt+A". It is also possible to capture specific character inputs by prefixing the character with "Char", e.g. "Charx" for an unmodified keypress on "x" or "CharX" for a Shift+X keypress. Non-letter key presses are available under their DOM event specified name, when available. These include Enter, Space, Tab, Delete, End, Help, Home, Insert, PageDown, PageUp, ArrowDown, ArrowUp, ArrowLeft, ArrowRight, Escape, ScrollLock, Pause, AltLeft, AltRight, ShiftLeft, ShiftRight, ControlLeft, ControlRight, MetaLeft, MetaRight, ContextMenu, F1-F24.

In addition to the keyboard events there are a number of events that are fired for a view that are not component specific.

Name	Description
Opened	A view has been opened and the components are ready for interation from users and process code.
Closing	The view is about to close down.
Close	The view has closed.
Focus	The view has gained focus.
Blur	The view has lost focus.
Maximize	The view size is maximized.
Restore	The view size is restored from maximized size.
ResizeEnd	The user has stopped resizing the view.
Minimize	The view is "minimized", e.g. effectively hidden until restored.
RestoreAppWin	The view is restored from minimized state.

Table: View Events

Since these events do not originate from any component, no component attribute should be supplied in the element.

Observe that the Unload events is triggered when the user is closing the browser.

Name	Type	Values	Description
view	string	Any string value	The name of the view from which the event will originate
event	string	Any event name	The event made by the user or the system
step	string	identifier	Step to execute

Table: Required attributes

Syntax

```

        <trigger view="[string]" event="[enumeration]" step="[string]"/>
```

Examples

```

        <?xml version="1.0"?>
<process name="myprocess" description="some description" author="XIOS/3">
  <trigger view="calculator" event="Alt+A" step="Add"/>
  <trigger view="calculator" event="Ctrl+B" step="Bold"/>
  <trigger view="calculator" event="Opened" step="Config"/>

  <step id="Add" name="Perform addition">
    ...
  </step>

  <step id="Bold" name="Make numbers bold">
    ...
  </step>

  <step id="Config" name="Configure settings">
   ...
  </step>
</process>
```

Reserved Events

These events are captured by XIOS/3 core.

Event	Description
CTRL+ALT+D	Toggle debug log
ALT+R	Send Refresh event to currrent component
CTRL+ALT+R	Reload current view
CTRL+ALT+Q	Close current view
CTRL+ALT+T	Open the Shell Application (terminal)

Table: Core Key Events

These events are captured by the current component, if there is one and the component is listening for the event

Event	Description

CTRL+A	Select all
Tab	Move focus to the next component
Enter	Confirm
Escape	Cancel
ArrowLeft	Move left
ArrowRight	Move right
ArrowDown	Move down
PageDown	Mov up
Delete	Delete
Home	Move home
End	Move to end
CTRL+C	Copy
CTRL+V	Paste
CTRL+X	Cut
CTRL+Y	Redo
CTRL+Z	Undo

Table: Component Key Events

System Events

System Events are events made by the XIOS/3 system. These can also be captured by the Process XML event listeners.

These events do not require any component or view attributes when defining them in an event listener. It does however need a communicator attribute to define which part of XIOS/3 throws the event.

The standard communicator is xios. You can however make you own communicator using the channel operation. Groups can also be communicators; when applications are stored in the groups virtual disk.

When a new folder is created by the filesystem the system will throw a "NewFolder" event.

Name	Type	Values	Description
communicator	string	Any string value	Communicator that is throwing the event
event	enumeration	Join, Saving, Completed, Login, LoginFailure, Authorized, Error, Close, Timeout, NewFolder, Insert	The event made by the system
step	string	identifier	Step to execute

Table: Required attributes

Syntax

```
<trigger communicator="[string]" event="[enumeration]" step="[string]"/>
```

Examples

```
<process name="xiosLogin" description="Main process file for Calculator" author="XIOS/3">
   <trigger communicator="xios" event="Login" step="Config"/>

   <step id="Config" name="Initiate Configuration">
     ...
   </step>
</process>
```

Using Expressions

This section provides you with an in depth coverage of the XIOS/3 Expression Language, which is used primary to allow for communication between data and components. In the chapter entitled "Definition" we explain the primary usage of the Expression Language and its importance in the Process XML language. We provide you with a series of easy to understand examples.

As a developer you are also allowed to use functions in the Expression language, a list of these can be found in the Functions chapter.

Definition

XIOS/3 uses an expression language to allow for communication between components and data. The language is simple but powerful and is a very important part of application logic.

The syntax of an expression can be divided in to two parts. The first part is a reference to XML data and can appear in several different ways. It can be a reference to a specific component, a variable containing data or a context data selection all of which we will explain in greater detail later. The second part of an Expression is an XPath expression which can be used to select data which is available in the first part of the Expression. In order to oversimplify we can say that the first part contains data while the second part, which is optional, is used to filter the data. Take a look at this expression:

#Explorer#treeFolders

The first expression only points to a component named treeFolders which is located inside a view named Explorer. Here we establish a fundamental rule, which goes "when referring to a component we must also specify the name of the view hosting that component". Basically we need the full address of the component. Knowing this will help you better understand the Process XML language in its entirety.

To the treeFolders component, which is a tree component, we bind XML data of a folder structure (example 1). If no selection has been made in that component this Expression reference will return the entire XML document, since the selection is by default is set to the root of the document. If a selection has been made (example 2); for example the first child element then it and its children will be available instead. If the selection had been made on the Pictures folder instead of the folder named home it would return the data in example 3.

Example 1:

```
<folders name="xios" id="100">
  <folder name="Home" id="200">
    <folder name="Document" id="201"/>
    <folder name="Pictures" id="202"/>
  </folder>
  <folder name="Applications" id="300"/>
</folders>
```

Selection: /folder[1] or / (root)

Example 2:

```
<folder name="Home" id="200">
  <folder name="Document" id="201"/>
  <folder name="Pictures" id="202"/>
</folder>
```

Selection: /folder[1]/folder[1] (first child of root)

Example 3:

```
<folder name="Pictures" id="202"/>
```

Selection: /folder[1]/folder[1]/folder[2] (second child of the first child of root)

If we instead would have used this Expression;

#Explorer#treeFolders#@name

In this Expression we use both parts; data and XPath. The second part of an Expression always starts with a hash-sign (#). Here we request the name attribute of the selected element. To continue the examples above is would result in the following:

Example 1: xios

Example 2: Home

Example 3: Pictures

As you can see only a text value would be available compared to XML data when we did not append the XPath expression for selecting the name attribute.

Variables and context data selection

As mentioned earlier in this text, the first part of an Expression can be presented in the form of a variable or in the form of context data. These might look like this:

#view1#component#XPath-Expression

Alias#XPath-Expression

!#XPath-Expression

#help#grid#@name

$treeData#@name

!#@name

Here we assume that the variable $treeData contains the same information as #Explorer#treeFolders. Also note that a hash-sign is used to separate the data from the XPath, just like in the previous examples. In the case of context data we use the exclamation mark to obtain the last selected data. When no selection has been made the exclamation mark will not contain anything. The scenarios above would result in this when using the Expression !#@name.

Example 1: (nothing)

Example 2: Home

Example 3: Pictures

Converting the Expression in to a string

Expressions can be converted in to string by placing them between curly brackets. It is good practice to do so whenever you are expecting a string value to be passed from your Expression. This makes it possible to aggregate several Expressions to form a large string, or combine it with text to do the same.

Expression can also be aggregated to form a string when combined with other text and placed inside curly brackets that will convert the value in to a string.

Name: {#Explorer#treeFolders#@name} ID: {#Explorer#treeFolders#@id}.

Example 1: Name: xios ID: 100.

Example 2: Name: Home ID: 200.

Example 3: Name: Pictures ID: 202.

External XML files

Expressions can also be used in conjunction with external XML data documents. Since the first part of an Expression is data we can therefore just replace the reference to the component and substitute it with an URL to a XML document like such;

http://www.somesite.com/documents/myfile.xml#/document[1]/chapter[1]/@title

In this expression the data is the XML contained in the file that the URL points to. The second part of the Expression is trying to extract the title of the first chapter. As you can see the XPath expression starts with a slash, which means that it starts at the root element.

Additional features in the Expression language

The Expression language also supports the usage of methods which can be applied on the data.

{#view1#component1.getvalue()}

Here we apply the getValue method on a component called component1 which is located in the view1. We also use curly brackets to convert the value in to a string.

Using Operations

Operations are building blocks of the application logic. They define that will happen when the end-user clicks on a specific component (together with the trigger element and the step element). Operations are logical command that will perform actions on components.

All operations must be placed inside a step element.

In this chapter every operation begins with an explanation about the operation and what it is used for. This is proceeded by a syntax and examples section, and then finally a related topics section.

Attributes

There is no specific syntax for operation element because different operations can have different attributes, thus different syntax.

Attributes

Required attributes

Name	Type	Values	Description
name	string	Any string value	The operation to use
value	string	string or expression	The data on which the operation should be performed. This value will always be evaluated

Examples

This section acts as a directory for examples. We have gathered many examples in the form of Applications, Components, Operations, Widgets and Layout.

Applications

This section contains detailed examples of applications developed for the XIOS/3 platform. They range from basic applications with limited functionality to more advanced and flexible applications.

Each application example is divided in to six parts; the introduction, The Application, The XML data, The UI XML, the The Process XML and Final Word. Together they cover code examples, descriptions, detailed information and images.

Updates

This part of the documentation will be continuously updated. You should revisit this section often for more application examples.

Book List

Here we demonstrate how to build a simple application (BookList) made up of a single view that lists books and allows you to select books and add new books to the list. This example shows the XIOS/3 MVC pattern XML structure in practice and clarifies how the model, view and control relate to one another.

The Application

As you can see below the application is very simple with only a few components. There is also some limited ability tochange the XML data. The focus is the list component which displays and handles the XML data.

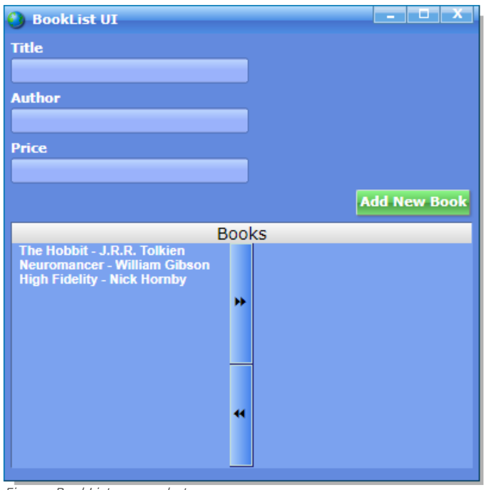

BookList UI
Title
Author
Price
Add New Book
Books
The Hobbit - J.R.R. Tolkien
Neuromancer - William Gibson
High Fidelity - Nick Hornby

Figure: BookList screenshot

The XML Data

The following XML data describes a list of three books. The book titles are described as attribute values and the author and price elements hold text values.

```
<?xml version="1.0"?>
<lists>
  <list name="Books">
    <book title="The Hobbit">
      <author>J.R.R. Tolkien</author>
      <price>20</price>
    </book>

    <book title="Neuromancer">
      <author>William Gibson</author>
      <price>10</price>
    </book>

    <book title="High Fidelity">
      <author>Nick Hornby</author>
      <price>15</price>
    </book>
  </list>
</lists>
```

Example: BookList data XML

The Application Package

The following XML describes the application package. It limits the number of instances to one and also defines what resources should be downloaded when the applications is taken offline. Both the view file and the process file will start onLoad.

```
<?xml version="1.0"?>

<application name="BookListApp" icon="xios/icons/network/32x32/earth.png" xmlns:xlink="http://www.w3.org/1999/xlink" instances="1">
  <settings>
    <path>examples/applications/</path>
    <home>home://Applications/</home>
  </settings>

  <about loader="false" version="1.0" revised="2019-01-28" created="2019-01-28" copyright="Copyright © 2019 XIOS/3 AB. All rights reserved.">
    <description>This is an example application of a booklist</description>
    <license xlink:type="simple" xlink:actuate="onRequest" xlink:href="https://xios3.com/license/xios_1.xml">XIOS.1</license>
    <installed>2019-01-28</installed>
  </about>

  <resources>
    <item xlink:href="examples/data/books.xml"/>
  </resources>

  <view xlink:actuate="onLoad" xlink:type="simple" xlink:href="examples/applications/views/booklist.xml"/>
  <process xlink:actuate="onLoad" xlink:type="simple" xlink:href="examples/applications/processes/booklist.xml"/>
</application>
```

Example: BookList data XML

The UI XML

The following UI XML describes the GUI of the BookList example application, presenting a book list application view. The "BookList screenshot" above, shows how the UI XML is visualized as an application.

The UI XML contains components for adding new book entries holding title, author and price fields and a button for adding the new entry.

The application view also lists all the existing books defined in the data XML above and enables the end-user to move book entries between the left and right parts of the list changing reading status of the books using the arrows in the middle.

```
<?xml version="1.0"?>
<view name="BookListView" title="BookList UI" version="1.0" height="400" width="400" icon="xios/icons/network/16x16/earth.png">
  <panel name="main" type="row" look="plate" width="auto" padding="5">
    <input name="title" label="Title" width="200" tabindex="1"/>
    <input name="author" label="Author" width="200" tabindex="2"/>
    <input name="price" label="Price" width="200" tabindex="3"/>
    <button name="addBook" text="Add New Book" width="100" tabindex="4" align="right"/>

    <list name="books" header="Books" mode="columns" width="auto" height="207">
      <item match="book" display="concat(@title,' - ',author)"/>
    </list>
  </panel>
 </view>
```

Example: BookList view (UI XML)

The process XML

The following process XML describes the programming logic of the BookList example application, connecting the above data XML and UI XML.

The process XML contains programming logic which:

- Starts the application view;
- Binds the book data to the list component in the view;
- Enables the interaction when adding new book entries

```
<?xml version="1.0"?>

<process name="BookListControl" description="BookList Process">
  <trigger view="BookListView" component="addBook" event="Select" step="3"/>

  <step id="1" name="Show Application view">
    <operation name="call" value="2"/>
  </step>

  <step id="2" name="Bind data">
    <alias name="dataDoc" value="examples/data/books.xml" model="value"/>

    <!-- Bind data -->
    <operation name="bind" value="$dataDoc">
      <component view="BookListView" name="books" select="/"/>
    </operation>
  </step>

  <step id="3" name="New Entry">
    <!-- add a new book entry to list -->
    <operation name="change" value="$dataDoc">
      <store select="/lists/list" type="append" model="data">
        <book title="{#BookListView#title}">
          <author>{#BookListView#author}</author>
          <price>{#BookListView#price}</price>
        </book>
      </store>
    </operation>

    <!-- set selection to last added item -->
    <operation name="setSelection">
      <component view="BookListView" name="books">
        <item select="/lists/list/item[last()]"/>
      </component>
    </operation>

    <!-- Set focus to title input field -->
    <operation name="callMethod" value="#BookListView#title">
      <method name="activate"/>
    </operation>

    <!-- Clear title, author and price input fields -->
    <operation name="action" value="#BookListView">
```

```
44        <component name="title" action="clear"/>
45        <component name="author" action="clear"/>
46        <component name="price" action="clear"/>
47      </operation>
48    </step>
49  </process>
```

Example: BookList process XML

Line 4

The trigger in the BookList process XML waits until a Select event occurs and will then call step 3 and start executing the operations in that step. The trigger action is initiated when the end-user clicks on the button named addBook in the application view.

Line 6-8

In step 1, the operation Open executes. This will display GUI of the application in the form of the event listenersUI XML file. The application should look like the screenshot in the "BookList screen shot" above.

Line 10-17

In step 2, an alias variable is assigned to the data XML document. The bind operation populates the list component named books in the BookList application view, using the data in the alias holding the data XML. This results in the list component showing the list of books, as seen in the "BookList screenshot" above.

Line 19-49

Step 3 is initiated by the trigger described above. First, the change operation executes. It appends a new book entry to the book list, using the text information in the Input fields named title, author and price filled in by the end-user via the GUI.

The expression enclosed in curly brackets may be unfamiliar to someone new to developing XIOS/3 applications. It will be covered in detail in the section Process Development. For now, you only need to know that the curly bracket expression will be replaced with the value of the input field it is referring to, which in this case is the text the end-user typed in the GUI input fields.

The second operation to execute is the setSelection operation, which displays the newly added book entry in the GUI as a highlighted entry in the book list.

Next, the input field named title will be focused using the method activate and made ready for new input by an operation which sets the cursor in the input field.

The last operation will Inputclear the fields named title, author and price, making them ready for new input.

Final Word

This application is very basic and covers very basics of XIOS/3 application development. The main focus was the logic (Process XML) and the simplicity of UI XML.

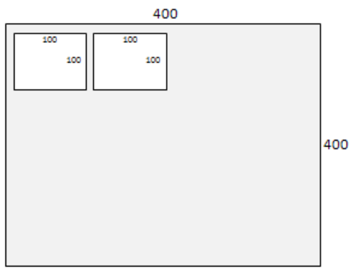
Layout

Layout examples

Column Layouts

Simple column layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

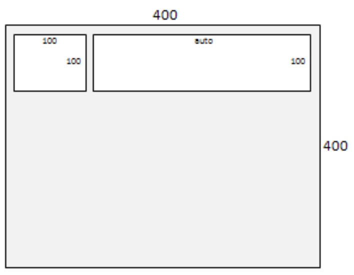
Column 1



```
<panel type="column" width="400" height="400" padding="5">
  <panel width="100" height="100" look="white"/>
  <panel width="100" height="100" look="white"/>
</panel>
```

Column 2

Simple column layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

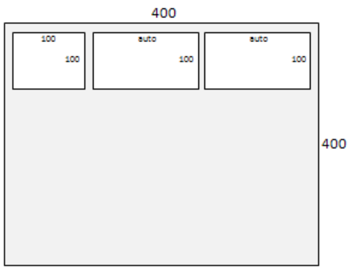


```
<panel type="column" width="400" height="400" padding="5">
```

```
  <panel width="100" height="100" look="white"/>
  <panel width="auto" height="100" look="white"/>
</panel>
```

Column 3

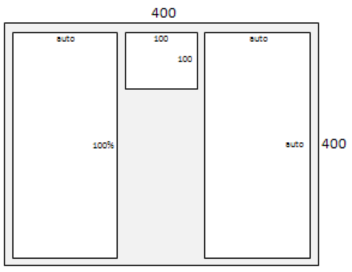
Simple column layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.



```
<panel type="column" width="400" height="400" padding="5">
  <panel width="100" height="100" look="white"/>
  <panel width="auto" height="100" look="white"/>
  <panel width="auto" height="100" look="white"/>
</panel>
```

Column 4

Simple column layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

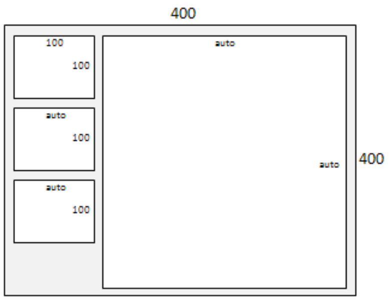


```
<panel type="column" width="400" height="400" padding="5">
  <panel width="auto" height="100%" look="white"/>
  <panel width="100" height="100" look="white"/>
  <panel width="auto" height="auto" look="white"/>
</panel>
```

Mixed Layouts

There are examples of mixed columns. The padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

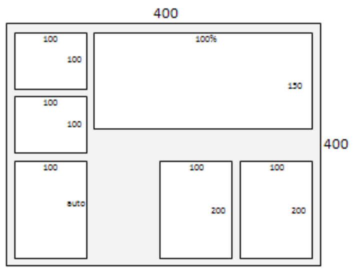
Mixed 1



```
<panel type="column" width="400" height="400">
  <panel type="row" width="100" height="100%" padding="5">
    <panel width="100" height="100" look="white"/>
    <panel width="auto" height="100" look="white"/>
    <panel width="auto" height="100" look="white"/>
  </panel>
  <panel type="column" width="auto" height="auto" padding="5">
    <panel width="auto" height="auto" look="white"/>
  </panel>
</panel>
```

Mixed 2

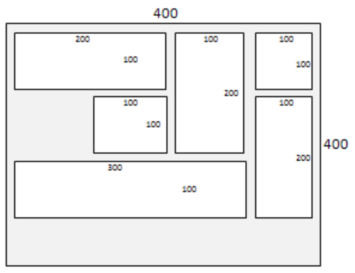
Mixed column/row layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.



```
<panel type="column" width="400" height="400">
  <panel type="row" width="100" height="100%" padding="5">
    <panel width="100" height="100" look="white"/>
    <panel width="100" height="100" look="white"/>
    <panel width="100" height="auto" look="white"/>
  </panel>
  <panel type="row" width="auto" height="100%" padding="5">
    <panel width="100%" height="150" look="white"/>
    <panel width="100%" height="auto" align="right" padding="5" valign="bottom">
      <panel width="100" height="200" look="white"/>
      <panel width="100" height="200" look="white"/>
    </panel>
  </panel>
</panel>
```

Mixed 3

Mixed column/row layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

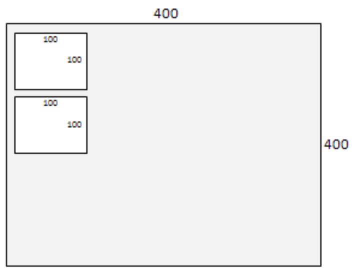


```
<panel type="column" width="400" height="400">
  <panel type="row" width="300" height="300" padding="5">
    <panel type="column" width="100%" height="200">
      <panel type="row" width="200" height="auto" padding="5">
        <panel width="auto" height="100" look="white"/>
        <panel type="row" width="auto" height="auto" align="right">
          <panel width="50%" height="auto" look="white"/>
        </panel>
      </panel>
      <panel width="100%" height="100%" look="white"/>
    </panel>
    <panel width="auto" height="auto" look="white"/>
  </panel>
  <panel type="row" width="auto" height="300" padding="5">
    <panel width="100%" height="100" look="white"/>
    <panel width="auto" height="auto" look="white"/>
  </panel>
</panel>
```

Row Layouts

Basic row layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.

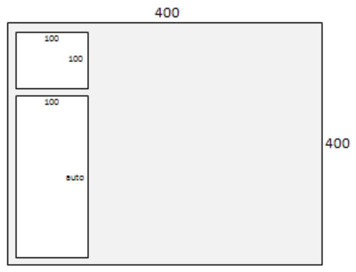
Row 1



```
<panel type="row" width="400" height="400" padding="5">
  <panel width="100" height="100" look="white"/>
  <panel width="100" height="100" look="white"/>
</panel>
```

Row 2

Basic row layout. Here the padding and look attributes are used to illustrate how the panels are positioned for copy-paste purposes.



```
<panel type="row" width="400" height="400" padding="5">
  <panel width="100" height="100" look="white"/>
  <panel width="100" height="auto" look="white"/>
</panel>
```

References

The References section contains a reference of all UI Components, ActionURLs, Process Operations and Functions that are available for developers.

The "Components" section provides you with in depth insight about components, their purpose, usage areas and also what actions, methods and events apply.

We also provide a complete reference to all logical operations available in the "Process Operations" section. For each operation we also have one or more examples.

ActionUrls

ActionURLs are functions that resemble in appearence to functions as they take input parameters but they are tied to a specific channel eg home:// and will perform the function on the defined channel.

lastLogins

An actionURL that lists the lastest logins to a specific group, providing an atom:feed document.

Syntax

The only parameter passed to this function is the group path (group/subgroup).

```
XML = [channel]://lastLogins([string])
```

Example

This example will provide the last logins of the group "XideZone" and the subgroup "europeans".

```
<alias name="lastLogins" value="home://lastLogins(XideZone/europeans)"/>
```

Related Topics

None available

leaveGroup

An actionURL that allow the user to leave a group.

Syntax

The only parameter passed to this function is the group path (group/subgroup).

```
nothing = [channel]://leaveGroup([string])
```

Example

This example will make the user leave from the "XideZone" subgroup "europeans".

Observe that when leaving a group the user looses any associations with that group (username, files etc).

```
<alias name="leave" value="home://leaveGroup(XideZone/europeans)"/>
```

Related Topics

None available

search

An actionURL that returns a composite file listing of search results matching the searhc query in the folder supplied and any subfolders.

This operation is very resource demanding when performing a recursive search. So it it wise to reconsider the usage of it.

Syntax

The first parameter is the root folder for the search. The second parameter is the serch term. The third parameter is the a flag if the search should be recursive for all child folders.

NOTE: always supply an end slash when refering to folder paths.

```
XML = [channel]://search('[string]','[string]','[boolean]')
```

Example

This example will produce an alias, $searchResults, which will contain an XML data document (atom:feed with atom:entry elements for each document found). The list will contain all documents which start with the characters 'abc' and the search will not be performed resursively in all the subfolders of 'home://Documents/'

```
<alias name="searchResults" value="home://search('home://Documents/','abc','false')" model="data"/>
```

Related Topics

None available

searchUser

An actionURL that returns an XML node-set of information about a user, including id and alias. A list of matching users is returned.

Syntax

This actionURL is very flexible and allows for input of several search variations. Two parameters can be passed. The first is required and is the search term, the second is the group path and is optional.

The first parameter performs a search for the alias (username) by default. If you with the perform a search on the user's id then write "user:[id]" instead.

If no group path is passed the the actionURL will perform a search among the xios group users. A group path might be found in the user's #GroupManager document.

```
XML = home://searchUser(['meta:value AND meta:value'],['grouppath'])
```

Example

In the below example we might know only the user's username but want to know the userId so we access it.

```
<alias name="userData" value="home://searchUser(goran)"/>
<alias name="userId" value="$userData#/atom:entry/dc:user"/>
```

Here we do the same but if we only know the userid but want to know the username.

```
<alias name="userData" value="home://searchUser(user:12884901950)"/>
<alias name="username" value="$userData#/atom:entry/atom:title"/>
```

Related Topics

None available

Components

Components are building blocks of an application and the visual representation of the possibilities of end-user interaction.

In this reference section every component is individually described, first what the component looks like and then how it is used. Depending on the usage area and the possibilities of the component is can have a short or very long usage section.

Component specific details are summarized in the details section. These include the type of element the component is, the component type, which element can be used as parent elements, what child elements are available and if it can use rules.

The attributes section is divided in to required and optional attributes and the bolded text represents the default value of the attribute.

For components that use specific child elements these are listed under the child elements section with their respective attribute. In the case that the component child elements have their own child elements, then these are also listed here, together with a reference of what the parent element is.

Actions, Events, and Methods, for every component is also listed in aplhabetical order. For each component a syntax and example section is also provided, together with links to examples and links to related components.

Accordion

This component defines a set of vertical sliding panels resembling an accordion.

Each Panel component has a header that shows the panel when clicked on. The accordion component is used to keep several panels in one column and provides a convenient way of switching between them. The component utilizes mutex.

Usage

This component is a container component that has Panel components as children and these panels can in turn contain other components such as Label or Input fields.

The accordion works if there are two or more panels inside of it, otherwise its main functionality cannor be used.

Details

Component Type: Navigation

Container Properties: High Level Element, Can act as a container

Parent Elements: Container, Panel

Child Elements: Panel

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px), percent (%)	The height of the component
width	measure values	integer (px), percent (%)	The width of the component

Optional Attributes

Name	Type	Values	Description
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
value	string	Any string value	Name of preselected panel

Child Elements

None available.

Actions

hide, show

Events

Init, Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<accordion name="accordion1" width="auto" height="auto">
<panel name="panel1" type="flow" look="white" selected="true" title="Panel 1" padding="5" icon="xios/icons/objects/16x16/brush1.png">
  <label name="lBrushes" default="Brushes"/>
</panel>
<panel name="panel2" type="flow" look="white" title="Panel 2" padding="5" icon="xios/icons/objects/16x16/palette.png">
  <button name="bOK" text="OK"/>
</panel>
</accordion>
```

Example: UI XML for the Accordion component

Related Components

Panel

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: Yes

Breadcrumbs

Creates a clickable path into a structure, like a path to a file.

Usage

The breadcrumb component uses rules to display the path to a selected node in a structure. The rule elements use the match attribute to select the ancestor nodes to display. Use the display attribute to create either output literals or data from the matched node. Note that any XPath used in the display attribute is relative to the rule match.

Details

Component type: Structure

Component Properties: Data Driven Component

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
height	measure values	integer (px), percent (%)	The height of the component. Defaults to 100%.
width	measure values	integer (px), percent (%)	The width of the component. Defaults to 25.

maxChars	numerical	positive integer	Maximum number of characters per label. Truncates with ellipsis.
defaultValue	string	Any string value	Default value to use if the selected node does not yield any text.

Child Elements

rule

The rule element is used to match specific data elements and display them as part of the path.

Required attributes

Name	Type	Values	Description
display	string	XPath Expression	XPath expression relative to the match attribute
match	string	XPath Expression	XPath expression relative to the data bound

Browser

Defines a web browser application component. It can be utilized to surf the web.

This component does not have any child elements. As for the render capabilities it uses the rendering engine of the browser that is currently browsing the OS.

Usage

Using the component is a matter of binding data to the browser component in the form of an URL. When the components starts to load the website it will make a Loading event and when it is finished it will make a Completed event.

Details

Component type: System

Component Properties: High Level Element, Data Driven component

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px), percent (%), auto, fill	The height of the component
width	measure values	integer (px), percent (%), auto, fill	The width of the component

Optional Attributes

Name	Type	Values	Description
href	string	about:blank, URL	Determines the startpage of the browser
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
value	string	about:blank, URL	Determines the startpage of the browser

Child Elements

None available.

Actions

hide, show

Events

Completed, Loading

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work. However in order to navigate to other webpages it needs to be bound to URLs.

```
<browser name="Browser1" width="auto" height="auto" href="http://www.xidezone.com"/>
```

Example: UI XML for the Browser component

Related Components

None available.

Button

Defines a clickable button with a text label. The button is the most common used component as regarding to interaction and comes with a design and interactivity functionality.

Usage

Buttons have very limited usage range. They are the basics or end-user interaction and are simply meant to be clicked. The component is controlled by attributes. Using the default attribute you can choose if the button should be selected by default to indicate an option that is preferred. By setting the default attribute to true it will be rendered as selected.

Once a button has been clicked it will fire a Select event.

The text attribute is used to create a label on the button. While the type attribute can if set to do or revert create the icons inside the button, illustrating a right pointing arrow (do) or a left pointing arrow (left). This can be used for undo/redo type functionality in applications in conjuntion with the disable action.

Details

Component Type: Selector

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component.
height	measure values	integer (px), percent (%)	The height of the component
text	string	Any string value	Creates a text label on the button
width	measure values	integer (px), percent (%)	The width of the component

Optional Attributes

Name	Type	Values	Description
btstyle	string	CSS Syntax	Overrides the default button text style
default	boolean	true, **false**	Determines if the button is selected by default
hide	boolean	**false**, true	Defines if the component should be hidden upon start up
enabled	boolean	false, **true**	Defines if the component should be enabled upon start up
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, do, revert	Sets the GUI type of the button

Child Elements

None available.

Actions

enable, disable, hide, select, show

Events

Select

Methods

setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<button name="button" text="Edit" width="100" height="20"/>
```

Example: UI XML for the button component

Related Components

None available.

ButtonBox

Defines a button looking box located inside its parent component. It can be displayed as a text and/or icon. The component has the same basic functions as the Button component but has several advantages since you can add you own icons inside the component.

Usage

The main purpose for this component is to aid the developer when making menus. It allows for more flexibility in the area of design than a normal button and can be incorporated in the ToolBar component.

In order to use the buttonbox component you need to set either the text or icon attributes. If both are set they will both appear next to each other with the icon the left and the text to the right. The text will only be displayed when the component is inside a toolbar or menu.

The mode attribute controls the behavior of the component. Setting the mode to switch will leave the buttonbox pressed in after the end-user has clicked on the component, as opposed to the default behavior, when mode is rebound, where the state of the component resets after the click.

ButtonBoxes can also utilize mutex behavior, by placing several buttonboxes inside a Group component that has the mode attribute set to mutex. The Group component offers several modes for buttonboxes.

When the selected attribute is set to true the buttonbox will be selected (pushed down) by default. Buttonboxes can also open Menu components using the submenu attribute by providing the name of the menu component.

Details

Component type: Selector

Component Properties: Child Element, Acts as menu button in toolbar

Parent Elements: Container, Group, Menu, ToolBar

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
height	measure values	integer (px)	The height of the component
icon	string	URL	Path to icon to show in the buttonbox
name	string	unique name	The name of the component
text	string	Any string value	Text displayed in the buttonbox
width	measure values	integer (px)	The width of the component

Optional Attributes

Name	Type	Values	Description
animate	boolean	**false**, true	
disabledIcon	string	URL	Path to icon displayed when the component is disabled
duration	measure value	milliseconds	The duration of the animation
enabled	boolean	**true**, false	Set to false the component will be grayed out
hide	boolean	**false**, false	Defines if the component should be hidden upon start up
hover	string	URL	Path to icon displayed when the component is hovered
icon2	string	URL	Path to icon to display when using animation
mode	enumeration	**default**, toggle	Decides how the buttonbox should behave when clicked
repeat	integer	[0...n]	How many times the animation should be repeated
selected	boolean	true, **false**	Indicates if the buttonbox is selected
style	string	CSS Syntax	Overrides the component's default style
submenu	string	Any string value	Name of the menu component to open
textposition	enumeration	**default**, ontop	Defines label position
title	string	Any string value	Tool-tip text
value	string	Any string value	Trigger value for button, if data is equal to the trigger, the button will be selected
iconsize	numeric	An icon size, 16, 32, 48 and 64 supported	Scales the supplied icon image into the specified size

Child Elements

None available.

Actions

enable, disable, hide, select, show

Events

ContextMenu, Hover, Release, Select

Methods

getValue, release, setIcon

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<toolbar name="fileToolbar" position="top" draggable="false">
<group name="fileGroup">
  <buttonbox name="bbNew" icon="xios/icons/applications/24x24/document_new.png" text="New" title="New"/>
  <buttonbox name="bbEdit" icon="xios/icons/applications/24x24/document_edit.png" title="Edit" mode="toggle"/>
  <buttonbox name="bbDelete" icon="xios/icons/applications/24x24/document_delete.png" text="Delete" title="Delete"/>
```

```
</group>
</toolbar>
```

Example: UI XML for the ButtonBox component

Related Components

Container, Group, Menu, ToolBar

Calendar

Defines a visually flexible and highly interactive calendar component that can be nagivated by week or month. It is used to visually display appointment data.

Usage

This component depends on data work correct. It must be bound to a data document, even if it does not contain any entries. Calendar has strict requirements regarding the construction of the data bound to the component. This is because it needs to be compatible with several different calendars.

Primarely, XML documents are intended to act as saved appointments. So document meta data from file listings are used to display appointments. If bound to a folder listing the component will work due to the fact that the folder listing, even if empty still returns XML.

The UI XML for the component is quite short, and most of the attributes are optional. Omitting the starttime and endtime attributes will cause the time range of the component to be 0-24. If however only business hours should be shown, say 07-16 then the starttime should be set to 07 and the endtime should be set to 16.

Type is the attribute that controls the visual mode of the component. Here the default value if 5week, which means that the calendar will be displayed as a week with five days. It can also be set to 7week to be displayed in 7 day mode. Day, today and month are also supported.

It is possible to make the component use the system calendar by setting the shared attribute to true. The component will then share behavior with all other components that have their shared attribute set to true, even if they are not part of the application.

Calendar has several standardized formats. Date is displayed as: YYYY-MM-DD, while time is displayed as hh:mm:ss. A full timestamp thus looks like this: YYYY-MM-DDThh:mm:ss(Z/+/-)hh:mm. Example: 2019-02-13T16:48:20Z+01:00 (Z is GMT time. +01:00 indicates 1 hour after GMT)

Keyboard navigation using the navigation keys (up, down, right, left) is possible. It can be combined with other view events such as Delete, ESCAPE and Copy.

Observe that some events are available only when making them on appointments (or rather data), these include ContextMenu and Select. Some events on the other hand are only available on the component it self (not directly on appointments), these include Menu.

Several methods are also offered to extract information and well as customize some attribute defined behavior. The setWeekStartDay method is used to define if the week should start on a Monday or Sunday and setRange is used to define how many hours should be displayed by the calendar component.

Selecting several cells simultaneously using the mouse is possible by capturing the Release event and then extracting the selected date and time using the following methods: getEndDate, getEndTime, getStartDate and getStartTime

Tip: Since filenames in folders can never the de same the need for unique file names emerges. For this the $tm system alias can be used since it generates a 32-bit integer. Since several files cant be created on the same millisecond.

Meta data standard

```
<atom:entry>
  <atom:title>[string]</atom:title>
  <atom:author><atom:name>[string]</atom:name></atom:author>
  <dc:subject>[string]</dc:subject>
  <dc:categories>[string]</dc:categories>
  <dc:startDate>[date]</dc:startDate>
  <dc:startTime>[time]</dc:startTime>
  <dc:startTimezone>[time]</dc:startTimezone>
  <dc:endDate>[date]</dc:endDate>
  <dc:endTime>[time]</dc:endTime>
  <dc:endTimezone>[time]</dc:endTimezone>
  <dc:location[string]></dc:location>
</atom:entry>
```

XML document file standard

```
<vevent>
  <uid/>
  <organizer/>
  <summary>[string]</summary>
  <dtstamp/>
  <dtstart>[timestamp]</dtstart>
  <dtend>[timestamp]</dtend>
  <description>[string|node-set]</description>
  <location/>
</vevent>
```

Details

Component Type: Selector

Component Properties: High Level Element, Data Driven Component

Parent Elements: Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
height	measure values	percent (%), integer (px)	The height of the component
name	string	unique name	The name of the component
width	measure values	percent (%), integer (px)	The width of the component

Optional Attributes

Name	Type	Values	Description
endTime	numerical	0-24	End time to display in calendar view
intervals	numerical	1-60	Number or time intervals an hour should be divided into when displayed
shared	boolean	**false**, true	If component should share calendar with system calendar
startTime	numerical	0-24	Starting time to display in calendar view
storeformat	enumeration	YYYY-MM-DD, YYYY/MM/DD, DD/MM/YYYY	Format of saved date
type	enumeration	**5week**, 7week, day, today, month	Calendar type to display
shorthand	boolean	**true**, false	Support appointment shorthand where clicking a day or making a range selection opens a small popup to enter appointment subject.

Child Elements

None available.

Actions

enable, disable

Events

ContextMenu, DateChanged, DateTime, DoubleClick, Drop, Menu, Release, Return, Select, SelectedDate, SelectedMore, SubjectMore

Special Events

Pressing **F2** when an Calendar-event is selected will allow you to edit that specific event without the need to open an external window with the Calendar component's internal event editor.

Methods

appointmentColor, changeInterval, getEndDate, getEndTime, getPaste, getSelection, getStartDate, getStartTime, setRange, seType, setWeekStartDay

Syntax and Examples

```
<calendar name="Calendar" type="5week" width="100%" startTime="06" endTime="20"/>
```

Related Components

Clock, Date

CheckBox

Defines a checkbox that changes it state from checked to unchecked when clicked and then vice versa when clicked again.

Usage

The default attribute is used to set the default pair of values that the checkbox will understand. The default value if false which will make the component work with true/false, you can however also work with off/on by setting the value of default to off or 0/1 by setting the value to 0. The look attribute will change the visual appearance of the component.

The state of the checkbox(checked or unchecked) can be accessed using Expressions, and will return the value of the component as either true or false. Depending on if the CheckBox is checked or not.

You can also use the preLabel or postLabel attributes to add some description text to the CheckBox in front of or behind the component.

The state of the checkbox can be changed using the setValue method to change the state of the checkbox. This is done by passing the value "true" or "false" in as parameters.

Details

Component type: Selector

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
checked	string	true, **false**	Decides if the checkbox should be checked by default

default	string	**false**, off, 0	The default pair of boolean values to be used
enabled	boolean	**true**, false	If the component is available for interaction
hide	boolean	**false**, true	Defines if the component should be hidden upon start up
label	string	Any string value	The label text
look	enumeration	**default**, bold, boldwhite, white	Defines the visual appearance of the component
lstyle	string	CSS Syntax	Overrides the default label style
postLabel	string	Any string value	Text label placed behind the checkbox
preLabel	string	Any string value	Text label placed in front of the checkbox
selected	string	true, **false**	Decides if the checkbox should be checked by default
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

None available.

Actions

disable, enable, hide, show

Events

Release, Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<checkbox name="USD" width="100" height="12" postLabel="Pay with US Dollars!"/>
```

Example: UI XML for the CheckBox component

Related Components

None available.

Clock

Defines a clock component that shows an analog or digital clock, optionally date, calendar(s) and/or names day.

Usage

The clock component is defined to show time in different ways.

You can use it to show a clock, like the widget in your widget pane or show the current time in different cities across the globe. The latter is made possible by adding city child elements and adding a diff attribute (time difference).

By setting the calendar attribute to true the clock component will instead be rendered as a calendar. When this has been set it is possible to set the number of calendar months to be displayed by setting the calendars attribute. When the component is displayed as a calendar you can utilize the href attribute, to render the calendar using XSLT the way you want to.

The namesday attribute will if set to true display today's namesdays according to Swedish namesday standard.

As this component also support data bindings and rules when it is displayed as a calendar (see example below). Here the rules are used to create visible selections of the data supplied but can also be used to display current week and so forth.

Details

Component Type: Selector

Component Properties: High Level Element, Data Driven Component

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
calendar	boolean	true, **false**	Decide if component should be displayed as monthly calendar
calendars	numerical	[0-12]	Specifies how many calendar months to show in the calendar
date	boolean	**false**, true	Shows current date if set to true

height	measure values	integer (px), percent (%), auto	Height of component
href	string	URL	Changes the default renderer for displaying a calendar
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style
mode	enumeration	**digital**, analog	Decides what clock mode to use
namesday	boolean	**false**, true	Show current names day
selectdate	boolean	**false**, true	Should dates in calendar be selectable
shared	boolean	**false**, true	Defines if it should share the global system calendar instead of creating a new instance
style	string	CSS Syntax	Overrides the component's default style
summer	boolean	**false**, true	Adjust for daylight savings time
title	string	Any string value	Tool-tip text
width	measure values	integer (px), percent (%), auto, fill	Width of component
xyscale	numerical	value in percent	Scales the clock to the provided percent value

Child Elements

None available.

Actions

hide, show

Events

Select

Methods

checkNote, getDateValue, getValue, setSkin

Syntax and Examples

These examples work on the fly and does not need to be bound to data to work.

```
<clock name="clockAnalog" summer="true" mode="analog"/>
```

Example: UI XML for an analog clock component (like the clock widget)

```
<clock name="clockDigital" summer="true" mode="digital"/>
```

Example: UI XML for an digital clock component (like the system clock)

```
<clock name="clockNamesdays" namesday="true"/>
```

Example: UI XML to display namesdays

```
<clock name="calClock" calendars="2" width="200" shared="false" calendar="true"/>
```

Example: UI XML for to display 2 calendars

```
<clock name="calClock" calendars="1" width="200" shared="false" calendar="true">
<rule match="/atom:feed">
  <startDate>{dc:startDate}</startDate>
  <endDate>{dc:endDate}</endDate>
</rule>
</clock>
```

Example: UI XML for calendar with rules to create a visual selection of the current week

Related Components

Date

Color

Defines a visual color picker and/or an inputfield for entering colors in hexadecimal format and the picker.

Usage

The component is used to allow end-users to easily choose colors via a GUI interface. The color that is retrieved using Expressions is in hexdecimal format.

An input field connected with the color component can also be created using the input attribute. There is also limited support for controlling the preselected color, using the color attribute, and transparency should be supported, using the transparent attribute.

Details

Component Type: Selector

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
color	string / numerical	#[rrggbb]	Sets the default color
hide	boolean	**false**, true	Decides of the component should be hidden when initially loaded
height	measure values	integer (px), percent (%)	The height of the component
icon	string	URL	Path to icon to display in the color component
input	boolean	**false**, true	If a textfield for entering RGB color should be displayed
label	string	Any string value	The label text
look	enumeration	default, classic, graphite	The visual appearence of the color component
lstyle	string	CSS Syntax	Overrides the default label style using CSS
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
transparent	string	**true**, false	If transparent color should be enabled
value	string / numerical	#[rrggbb]	Sets the default color
width	measure values	integer (px), percent (%)	The width of the component

Child Elements

None available.

Actions

disable, enable, hide, show

Events

Return, Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<color name="colorExample" width="auto" height="auto" color="#ff0000"/>
```

Example: UI XML for the Color component

Related Components

None available.

ComboBox

Defines a component that displays and allows for selection of item values. It works similarly to a HTML select element with more functionality. It consists of two parts; the combobox component and the plate component (with its child elements). The latter contains all available values that can be selected.

Usage

Comboboxes can be populated by defining static child elements or by binding data to the component and populate it with data options that are dynamically updated.

By setting the editable attribute to true you can allow the end-user to insert values in to the combobox. Thus is can also act as an input field.

The subplate attribute is used when you have defined a plate element outside the combobox element, so that it no longer is a child element. This might be done for several reasons, such as two comboboxes share the same plate. To be able to do this you need to provide the name of the desired plate in the subplate attribute.

One item child can have the selected attribute, making it the pre selected value inside the combobox. When retrieving the value of the combobox it will be equivalent to the text() in the selected item child element, unless a value attribute it set on the child element then it will equal the value attribute.

The combobox can use rules to render child elements dynamically using bound data. In the example at the bottom, we use rules to match the component element, and then proceed to output an item element for every component in the XML data document.

The end-user can also interact with the comboxbox using the keyboard. Arrow up and down are used to nagivate the plate. Return us used set the value in the combobox and Escape will close the plate.

Details

Component Type: Selector

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: Plate

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%), auto, fill	The width of the component

Optional Attributes

Name	Type	Values	Description
placeholder	string	Any string value	Displays a pretext that will disappear as soon as a value is selected in the plate
editable	boolean	true, **false**	If the combobox should act as an input field
hide	boolean	**false**, true	Decides of the component should be hidden when initially loaded
label	string	Any string value	The label text
look	enumeration	default, classic, grid	Defines the UI of the component
lstyle	string	CSS Syntax	Overrides the default label style using CSS
style	string	CSS Syntax	Overrides the component's default style
subplate	string	Any string value	The name of the plate to display
title	string	Any string value	Tool-tip text
usethumbnail	boolean	**true**, false	If the combobox should display the selected value as visualied in the child element or with plain text

Child Elements

See Plate

Actions

enable, disable, hide, show

Events

Change, Focus, Select

Methods

clear, getValue, setValue

Syntax and Examples

The first example requires data to be bound to the plate so that the rules can display the desired items. The second example does not require data, because the items are predefined.

```
<combobox name="cbExample" width="200" height="20" editable="false">
  <plate name="myPlate" height="200">
    <item> Choose a component </item>
    <rule match="component">
      <item>{@name}</item>
    </rule>
  </plate>
</combobox>
```

Example: UI XML for the ComboBox component (data needed)

```
<components>
  <component name="Accordion" tag="Component" group="UI XML">
    <documentation>
      <description>Accordion description.</description>
    </documentation>
  </component>
  <component name="Browser" tag="Component" group="UI XML">
    <documentation>
      <description>Browser description.</description>
    </documentation>
  </component>
  <component name="Button" tag="Component" group="UI XML">
    <documentation>
      <description>Button description.</description>
    </documentation>
  </component>
</components>
```

Example: Data XML for the ComboBox component

```
<combobox name="cbExample" width="200" height="20" editable="false">
  <plate name="myPlate" height="200">
    <item>Combo Box</item>
    <item selected="true">Combination Box</item>
    <item>DropdownBox</item>
  </plate>
</combobox>
```

Example: UI XML for the ComboBox component (no data needed)

Related Components

None available.

Container

Defines a container that hosts other components described by its rules and dynamically creates and manages these components depending on the incoming data.

Usage

The main usage of this component comes when designing a GUI based on availble data. This is accomplished by using rules. In this case the rules will, when the match condition is met, render another component and populate it with data if so desired.

In the example at the bottom of this page we find a code snippet, that will create a text labels for every term element it finds. In the labels the value of the text attribute of the term element will be displayed. The value is set using Expressions to find the name attribute.

Using setSelection you can set define the selection that will be used in the container.

Details

Component Type: Placeholder

Component Properties: High Level Element, Data Driven Component

Parent Elements: Panel

Child Elements: None available

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
height	measure values	integer (px), percent (%), auto	The height of the component
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
scroll	enumeration	**auto**, scroll, visible, false	Controls the scroll behavior
show	enumeration	**default**, all, selected	Defines the visibility of the rendered components
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
width	measure values	integer (px), percent (%), auto	The width of the component

Child Elements

None available.

Actions

hide, show

Events

None available.

Methods

clearContent, getSelected

Syntax and Examples

Observe that this example needs to be bound to the data in order to work.

```
<container name="coButtons" show="all" width="300" height="100">
<rule match="component">
  <button text="{@name}" width="100"/>
</rule>
</container>
```

Example: UI XML for the Container component

Related Components

Info

DataItem

Defines a component that creates data-dependent menu items.

Usage

This component renders menu items dynamically depending on the rendering rules that are set.

The created item (of which there can be several) have all the attribute options as regular menu items.

Using setSelection you can set define the selection that will be used in the dataitem.

Details

Component Type: Atomic

Component Properties: Child Element, Data Driven Component

Parent Elements: Menu

Child Elements: None available.

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component
height	measure values	integer (px), percent (%)	The height of the component

Optional Attributes

Name	Type	Values	Description
draggable	boolean	**false**, true	Defines if the generated options should be draggable
enabled	boolean	**true**, false	Set to false the items will be grayed out
sort	boolean	**false**, true	Defines if the items should be sorted alphabetically
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

None available.

Actions

enable, disable, hide, show

Events

Select

Methods

None available.

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work.

```
<menubar name="menuBar" draggable="false" position="top" opacity="50">
<menu name="componentMenu" text="Components">
  <dataitem name="componentDynamicMenu">
    <rule match="components/component">
      <item icon="xios/icons/development/16x16/components.png" text="{@name}"/>
    </rule>
  </dataitem>
</menu>
</menubar>
```

Example: UI XML for the DataItem component

Related Components

Item, Menu, Menubar

Date

Defines a date component that is used to format and display date values. The component will render an input field and a small icon; which will when clicked launch a date picker tool.

Usage

When a date is selected in the date picker it will appear inside the input field. The selected value is then the value of the component. The value attribute is formatted according to ISO 8601 date order (YYYY-MM-DD) standard. You can insert a predefined static value or use the today() value to extract current date for every individual end-user, the latter is done automatically.

The end-user can change this format by changing the value in #Settings (/settings/date), which can be done using the Control Panel application in the "Locale" section, while in the "Date & Time" tab. The date is always presented as requested by the end-user. The developer will however always work in the ISO 8601 standard.

By using the icon attribute you can choose where the icon should be located in relation to the input field.

It is possible to make the component use the system calendar by setting the shared attribute to true. The component will then share behavior with all other components that have their shared attribute set to true, even if they are not part of the application.

Details

Component type: Selector

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
hide	boolean	**false**, true	Decides of the component should be hidden when initially loaded
label	string	Any string value	The label text
look	enumeration	**default**, graphite	The visual appearence of the component
lstyle	string	CSS Syntax	Overrides the default label style using CSS
shared	boolean	**false**, true	If component should share calendar with system calendar
style	string	CSS Syntax	Overrides the component's default style
storeformat	enumeration	YYYY-MM-DD, YYYY/MM/DD, DD/MM/YYYY	Format of saved date
title	string	Any string value	Tool-tip text
toggle	boolean	**false**, true	If the calendar picker should be toggled when the the input field component is clicked
value	numerical	**default**, [yyyy-mm-dd], today()	The date currently showing

Child Elements

None available.

Actions

disable, enable, hide, show

Events

DateChanged, Focus, Select

Methods

getValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<date name="dateToday" value="today()"/>
```

Example: UI XML for the Date component, with current date predefined

Related Components

Calendar, Clock

Divider

Defines a divider component, which is a simple component that is used to separate menu items. When used it will be displayed as a line beween menu items.

Usage

The purpose of the divider component is to divide menu options. End-users cannot interact with this component.

Details

Component Type: -

Component Properties: Child Element, Menu Component

Parent Elements: Menu

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

None available.

Actions

None available.

Events

None available.

Methods

None available.

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<menubar name="menuBar" draggable="false" position="top">
  <menu name="mFile" caption="false" text="File">
    <item name="mOpenDoc" text="Open" icon="xios/icons/applications/16x16/document_into.png"/>
    <divider name="md1"/>
    <item name="mNew" text="New" icon="xios/icons/applications/16x16/document_new.png"/>
  </menu>
</menubar>
```

Example: UI XML for the Divider component

Related Components

Item, Menu

Form

Defines a form component that is used to stack and popluate traditional form components such as Input, ComboBox and TextArea. If a more advanced form formatting is desired that can be accomplished by using Panels.

Usage

This component is guided by Rules and also needs data to be bind to function. The rules will only render a component of their match attribute is found.

Details

Component Type: Placeholder

Component Properties: High Level Element, Wrapper Component

Parent Elements: Panel

Child Elements: None available

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px), percent (%)	The height of the component
width	measure values	integer (px), percent (%)	The width of the component

Optional Attributes

Name	Type	Values	Description
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

None available.

Actions

hide, show

Events

None available.

Methods

None available.

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work.

```
<form name="formExample" vmargin="30" hmargin="10" height="auto">
<rule match="component/@tag">
  <input width="80"/>
</rule>

<rule match="component/@group">
  <combobox name="comboboxExample" width="200" height="20" editable="false">
```

```
    <plate name="plateGroups" height="200">
      <item>Application</item>
      <item>Data dependent</item>
      <item>Expander</item>
      <item>Interaction</item>
      <item>Layout</item>
      <item>Menu</item>
      <item>Presentation</item>
      <item>Side Menu</item>
      <item>Time</item>
      <item>Toolbar</item>
    </plate>
  </combobox>
</rule>
</form>
```

Example: UI XML for the Form component

Related Components

None available.

Grid

The grid component is used to present much data information in cell format. This is accomplished by defining a row and columns in which the data is populated.

Headers are displayed above every column and are also sortable (ascending or descending) by column values.

Usage

Basic grid understanding begins with understand how the grid is constructed, how it will be rendered and the rule that data must be bound to it.

As in the example at the bottom of this page we begin by defining the attributes of the grid element. They control how the grid will behave when the end-user interacts with it, how it will look and how updated data will be handled. In the example there are two child elements, row and rule. The row element is the only real child element that the grid has. The rule however is not really a child element, it is more of an extra markup element to modify the row element above it. Here the row matches an element called component from the data document. The row itself has four column element which themselves match parts of the data element (e.g. component/@name). The fourth column will match documentation/description and it is here our rule apply, which matches the same data. So when this condition has been met the grid will render a label with the value of cell.

Looking at the grid as a whole and the attributes available we find the datachange attribute. When this is set to true it will update the value in its cells when the data document that it is bound to has been updated by a data transaction. If it is set to false no updates will be made.

The inset attribute will make the grid looked like it is sunken in (embedded) the layout.

The mode attribute controls the behavior of the grid, where the default value will allow for single cell selection. If the mode is set to grid it will instead render a numbered plate to the left of every row, showing the row number. Furthermore only row selection will be possible.

The selection attribute decides how a selection will be displayed and what data will be selected. When set to row an entire row will be selected. When set to cell single cells will be selectable, unless the mode attribute is set to grid. If you don't use the attribute the values will not be selectable. Setting the draggable attribute to true will allow the rows to become draggable creating a simple drag and drop functionality.

The row element uses the match attribute to select what data will be available to its children, the column elements. These in turn also use the match attribute to select data. By using a combination of either attributes match and display or attributes match and look, the data will be correctly rendered. By using the look/match combination you can make cells editable. The display/match attribute makes cells uneditable.

The look attribute decides how the grid cells should look. This only applies if the display attribute has not been set. For normal cells you should set the value to classic.

Finally the sort and type attributes are used when presorting values. This is done by setting the sort attribute to either ascending (lowest to highest) or descending (highest to lowest) and the type attribute to either alpha (for sorting alphabetically) or numeric to sort by numbers.

By using the **name** attribute on the **column** child element, you will be able to listen to events made specifically from the row which the column belongs to. The rows can fire "Select" and "Change" events, which allow you to listen for them in Process XML.

Using setSelection you can set define the selection that will be used in the grid.

Details

Component type: Structure

Component Properties: High Level Element, Data Driven Component

Parent Elements: Container, Panel

Child Elements: row, column (child of row)

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component
height	measure values	integer (px), percent (%)	The height of the component

Optional Attributes

Name	Type	Values	Description
datachange	boolean	**true**, false	If the component should update upon data change
draggable	boolean	**false**, true	Makes the grid items draggable
inset	boolean	**false**, true	Make the grid look sunk in to the layout
label	string	Any string value	The label text
look	enumeration	default, graphite	The visual appearence of the grid
lstyle	string	CSS Syntax	Overrides the default label style using CSS

mode	enumeration	grid, **default**	The display mode of the grid
pageMessage	string	Any string value	Message to display in paginationbar when the component starts paginating the results
scroll	boolean	**true**, false	Controls scroll behavior
selection	enumeration	row, cell, **default**	Determine selection mode
style	string	CSS Syntax	Overrides the component's default style

Child Elements

row

The row element uses a match attribute to select nodes of the data.

Name	Type	Values	Description
height	measure values	integer (px)	Height in pixels
match	string	Xpath Expression	Required. XPath expression relative to the data bound

column (child element of row)

The column elements use a match attribute to select nodes of the data.

Name	Type	Values	Description
align	enumeration	**left**, right	Alignment of text inside column cells
display	string	match	Selection of data to display. XPath expression relative to the data bound to the match attribute
draggable	boolean	**true**, false	Decides if column should be draggable
filter	boolean	**false**, true	Allow filtering in column
icon	string	URL	Icon to display in column header in front of column label
label	string	Any string value	The label text
look	enumeration	**default**, classic, white	Defines the look of the cell
match	string	Xpath Expression	Required. XPath expression relative to the data bound to the parent row
name	string	unique name	Unique identifier for the column. This value is displayed in the column if no label is provided
scroll	boolean	**false**, true	Decides if scroll should be added to the cell if content length exceeds cell width
sort	string	**ascending**, descending	Order in which to sort. Sorting is case insensitive.
title	string	Any string value	Tool-tip text for column
type	string	**alpha**, numeric, integer, float, int, pubdate	Alphabetical or numerical sorting
width	measure values	integer (px), percent (%)	Width of the column.

Actions

disable, enable, hide, show

Events

ContextMenu, Drop, Delete, DoubleClick, Header, HeaderContextMenu, Return, Select

Methods

deselect, getValue, hideColumn, selectAll, selectFirstRow, selectLastRow, showColumn, sortColumn, update

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work.

```
<grid name="gridExample" width="auto" height="auto" selection="row" datachange="false" inset="false">
<row match="component" height="50">
  <column width="60" name="Name" match="@name" display="." />
  <column width="40" name="Tag" match="@tag" display="." />
  <column width="80" name="Group" match="@group" look="classic" />
  <column width="160" name="Description" match="documentation/description"/>
</row>

<rule match="documentation/description">
  <label style="font-weight: bold; font-family: verdana; font-size: 15px; padding-top: 2px;"/>
</rule>
</grid>
```

Example: UI XML for the Grid component

```
<components>
<component name="Accordion" tag="Component" group="UI XML">
  <documentation>
    <description>Accordion description</description>
  </documentation>
</component>
<component name="Browser" tag="Component" group="UI XML">
  <documentation>
    <description>Browser  description</description>
  </documentation>
</component>
<component name="Button" tag="Component" group="UI XML">
  <documentation>
```

```
    <description>Button description</description>
  </documentation>
</component>
</components>
```

Example: Data XML for the Grid component

Related Components

None available.

Group

Defines a group container component that will group several components and let them share behavior.

Usage

The Group component alows several similar components to share behavior, which is controlled by the mode attribute. The most important of these is the mutex mode which allows for only one component to be selected or visible at any given time. Some components such as radio buttons will (when grouped) exhibit this behavior regardless of if this attribute is set.

The same attribute can be set as toggle, which will make a ButtonBox appear as selected (active), while also using the mutex behavior. The switch mode will also make buttonboxes appear selected but does not have any mutex relationship which means that several buttons can be selected at the same time.

For layout purposes you can use the align attribute which will determines the alignment of all components within the group. There is also the hide attribute which, if set to true, will hide the group and all components it contains.

Details

Component Type: Placeholder

Component Properties: High Level Element, Wrapper Component

Parent Elements: Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
align	enumeration	right, **left**, center	Sets the alignment of the components that are located within the group
height	measure values	integer (px)	The height of the component
hide	boolean	true, **false**	If set as true the group will be hidden
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
mode	enumeration	**default**, mutex, switch, toogle	Sets the behavior mode of the group
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
value	string	Any string value	Preselected value or component inside group
width	measure values	integer (px)	The width of the component

Child Elements

None available.

Actions

enable, disable, hide, show

Events

Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<group name="group1" mode="mutex">
<radio name="radio1" postLabel="One" value="1"/>
<radio name="radio2" postLabel="Two" value="2"/>
<radio name="radio3" postLabel="Three" value="3"/>
</group>
```

Example: UI XML for the Group component

Related Components

None available.

Image

Defines a component that makes it possible to display an image. The supported types are GIF, PNG and JPEG format.

Usage

The image component works by simply adding a URL to an image in the src attribute. The image will be rendered in its full size, however you can use the scale attribute to make the image fit in to the component that you want to.

The image component has two layout attributes, the align and valign. Where align is the attribute to set the horizontal alignment of the image, and valign is the vertical alignment of the image.

If the scale attribute is set to true it will make the image scale to fit the defined size otherwise it will be displayed as the actual size. There is also the possibility to make the image a link by setting the type attribute to link.

By refereing another to another image in the hoversrc attribute you can change the image source when you hover your image component.

Details

Component Type: Atomic

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px)	The height of the image
src	string	URL	Path to the image to display
width	measure values	integer (px)	The width of the image

Optional Attributes

Name	Type	Values	Description
align	enumeration	right, **left**, center	Horizontal alignment
hide	boolean	**false**, true	Defines if the image should be hidden by default
hoversrc	string	URL	Path to the image to display while hovering
scale	boolean	true, **false**	Scale the image to fit width and height
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, link	Sets the image's behavior type
valign	enumeration	**top**, middle, bottom	Vertical alignment

Child Elements

None available.

Actions

hide, select, show

Events

ContextMenu, DoubleClick, Drop, Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<image name="imageSave" src="xios/icons/applications/48x48/disk_blue.png" title="Floppy Disk" align="right"/>
```

Example: UI XML for the Image component

Related Components

None available.

Input

Defines an single-line input field that is editable that can be styled several different ways.

Usage

The main purpose of this component is to allow end-users to insert information which later can be processed. This component is entirely controlled by attributes.

The enabled attribute is used to limit end-user interaction by graying out the input field, which is accomplished if the attribute is set to false. The default attribute displayes the default text inside the input field when no data is bound to it.

The focus attribute is used to focus (select) the input field. If several input fields have the attribute set to true, the last rendered will be focused. The length attribute is used to limits the number of characters that can be inserted.

This component allows for heavy styling using the look attribute. There are two three types of looks. First the default, which will create a blue input field, while white will create a white input field. Both of these have rounded corners. Observe that you can't apply custom styling on the input field.

The type attribute controls what type the input field should be. The default value will just render a normal input field, the value counter will render an input field with buttons to increase and decrease the value. A default value must be set or bound to the input field in order for it to work. The value password will hide the characters written and replace them with asterisks. The value new-password will work the same as password, but indicate to any password manager that they shouldn't attempt to pre-fill any values. When it is set as numeric it will only accept numbers, which makes it useful for inputting only numerical values. Note that this type will also allow you to define the maxValue and minValue attributes to set the upper and lower limit of the acceptable numeric value inside the input field. The default values are 9999 for maxValue and -9999 for minValue. You can also define the size of the increment by defining a number in the step attribute, which has the default value of 1, meaning that it will increase or decrease by one.

When using the icon attribute an inline button will be displayed that once clicked will fire the "Select" event.

Details

Component Type: Atomic

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component

Optional Attributes

Name	Type	Values	Description
placeholder	string	Any string value	A text visible in the input which is removed at first input
default	string	Any string value	Default text of input field of component
enabled	boolean	**true**, false	Set to false the component will be grayed out
focus	boolean	true, **false**	If the component should be focused
hold	boolean	true, **false**	If component should stop listening to data updates
icon	string	URL	Icon to display as an inline button in the input field
length	numerical	[0...n]	Sets the maximum length of the text string that can be entered
look	enumeration	**default**, graphite, blueold	Defines the look of the component
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
maxValue	numerical	0...n	Maximum number when numeric type
minValue	numerical	0...n	Minimum number when numeric type
mode	enumeration	**default**, text	If the inputted text should be handled as an encoded URI or plain text
step	numerical	[1...n]	The size of incrementation when in numeric mode
style	string	CSS Syntax	Overrides the component's default style
tabindex	numerical	1...[n]	Defines the tab order
title	string	Any string value	Tool-tip text
tstyle	string	CSS Syntax	Style for text in input field
type	enumeration	**default**, counter, password, new-password, numeric, selector, icon	The type of mode that should be used

Child Elements

None available.

Actions

enable, disable, hide, show

Events

Blur, Change, Escape, Focus, Key, Return, Select

Methods

activate, clear, getValue, setLabel, setValue

Syntax and Examples

These examples work on the fly and does not need to be bound to data to work.

```
<input name="inputName" width="100" default="Enter your name and press enter"/>
```

Example: UI XML for the Input component (Normal input field)

```
<input name="inputPassword" width="100" type="password"/>
```

Example: UI XML for the Input component (Password input field)

```
<input name="inputCounter" width="100" default="1" type="counter"/>
```

Example: UI XML for the Input component (Counter input field)

Related Components

None available.

Item

Defines items in lists, menus, plates.

- List Item
- Menu Item
- Plate Item

Related Components

List, Menu, MenuBar, Plate

List Item

Defines an item in the List component.

Usage

The match attribute in the item element defines what elements in the data XML that should be bound, and the display attribute defines what will be displayed of the list item using XPath expressions.

Details

Component Type: Atomic

Component Properties: Child Element

Parent Elements: List

Child Elements: No child elements

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
display	string	**@name**, Expression	Relative to the match attribute. The text displayed
match	string	@name, Expression	What data is to be selected

Optional Attributes

Name	Type	Values	Description
class	string	Any string value	CSS class for item default state
selectionClass	string	Any string value	CSS class for item select state
hoverClass	string	Any string value	CSS class for item hover state
icon	string	URL	Adds icon in front of row

Actions

None available.

Events

None available.

Methods

None available.

Syntax and Examples

See List component.

Menu Item

Defines static menu items in application menus.

Usage

Menu items can exhibit radio button behavior and checkbox behavior. To define which item should be selected when menu items are grouped, using the group attribute, you use the select action on the menu item.

You can also create checkable menu items by setting the checkable attribute to true. This will enable the menu item to act as a checkbox component. Observe that when you utilize the checkable or group behavior that you cannot use any icons, since the default icons for this will be used.

A menu item cannot use both the group attribute and the checkable attribute. If both are set the group attribute will have higher priority.

To create a submenu you should use the submenu attribute. The submenu attribute value should be set to the name of the menu component that you wish to appear as a submenu.

The **hotkeytext** attribute is used to display a secondary text in a menu item. It is designed primarely for hotkey commands, thus simply allowing the end-user to learn the keyboard hotkey for that specific action.

Details

Component Type: Atomic

Component Properties: Child Element

Parent Elements: Menu

Child Elements: No child elements

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
text	string	Any string value	Text to be shown in the menu

Optional Attributes

Name	Type	Values	Description
checkable	boolean	**false**, true	If the item should be checkable
checked	boolean	**false**, true	If the item should be prechecked
enabled	boolean	**true**, false	Set to false the component will be grayed out
group	string	Any string value	Name of group. Used when grouping items
hide	boolean	**false**, true	Decides if the component should be hidden
hotkeytext	string	Any string value	Secondary text used to indicate hotkey command.
icon	string	URL	Path to icon for the item
selected	boolean	**false**, true	If item is part of a group, have this item default selected
style	string	CSS Syntax	Overrides the component's default style
submenu	string	Any string value	Name of submenu to open when selecting item
title	string	Any string value	Tool-tip text

Actions

enable, disable, hide, show

Events

Release, Select

Methods

getValue, setIcon, setSelect, toggleCheckable

Syntax and Examples

```
<menubar name="menuBar" draggable="true" position="top">
<menu name="mFile" caption="false" text="File">
  <item name="mCheck" text="Checkable Item" checkable="true"/>
  <item name="mNew" text="New" icon="xios/icons/applications/16x16/document_new.png" submenu="mTools"/>
</menu>
</menubar>
<menu name="mTools">
<item name="mTool1" text="Tool 1"/>
<item name="mTool2" text="Tool 2"/>
</menu>

```

Example: Example for checkable menu items

See Menu component for additional examples.

Plate Item

Defines static items in the Plate component.

Usage

Item elements are single options available inside the plate. Each item element represents a single option. The element itself contains information about what the option should display.

Details

Component Type: Atomic

Component Properties: Child Element

Parent Elements: Plate

Child Elements: No child elements

Can Use Rules: No

Attributes

Required Attributes

No required attributes.

Optional Attributes

Name	Type	Values	Description
colorbox	string / numerical	#[rrggbb]	The color of the box icon
selected	boolean	**false**, true	Defines if the item should be selected
title	string	Any string value	Tooltip text
value	string	Any string value	The value of the selected option

Actions

None available.

Events

None available.

Methods

None available.

Syntax and Examples

See Plate component.

Label

Defines a text label, used mainly to display small or large amounts of text.

Usage

This component is used to create a label text. It has the ability to contain a default text until data is bound to the component and also to act as a link. The text is defined using the default attribute. The actual text can be dynamically added when binding data to the component, which will remove the text value previously displayed.

The label component offers several attributes to style the label. The bgcolor is used to set the background-color of label. The color attribute sets the color of the text, and the fgcolor attribute sets the text-color while the end-user is hovering over the component. There are also the bold attribute which can be used to make the label bolded, the font attribute white is used to select the font in which the text will be displayed. Finally the size attribute defines the size of the characters.

Labels can also acts as links, this is possible if the type attribute is set to link. This will cause the label to display a underlining when hovered. The underlining can be removed by setting the decoration attribute to none. Once your label has been defined as a link you can also style it. The hoverClass allows you to define the CSS class taht will be used when the end-user is hovering the component.

Using the setValue method and the format attribute the label component might display date and time related information. If the format attribute is set to **timestamp**, then setValue might be used to update the timestamp value. Observe that the the timestamp will only be updated the passed information will not be displayed.

Details

Component Type: Atomic

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
bold	boolean	true, **false**	If the text should be bolded
bgcolor	string, numerical	#[rrggbb]	The text's background color on hover
color	string, numerical	#[rrggbb]	The color of the text
decoration	enumeration	**default**, none	If an underlining should be displayed
default	string	Any string value	Default text to show, when no data is bound
disableClass	string	Any string value	CSS class used when the label is disabled

ellipsis	boolean	**true**, false	If ellipsis should be used when the text is cut off
enabled	boolean	**true**, false	If interaction should be possible with the component
fgcolor	string, numerical	#[rrggbb]	Sets the text color on hover
font	string	Any string value	Defines the font-family to use on the text
format	enumeration	**default**, timestamp, date, time	Defines what information should be displayed in the label
height	measure values	integer (px)	The height of the component
hide	boolean	**false**, true	Defines if the component should be hidden upon start up
hoverClass	string	Any string value	CSS class used when hovering over a link label
icon	string	URL	Defines the icon to use
iconPos	enumeration	**left**, right	Defines the position of the icon
iconsize	numerical	[0...n]	Sets the maximum size of the icon, in pixels.
label	string	Any string value	The label text
look	enumeration	**default**, bold, boldwhite, white	Graphical look of the label
lstyle	string	CSS Syntax	Overrides the default label style using CSS
size	numerical	[0...n]	Defines the size of the text
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, link	Sets the behavior of the label
value	string	Any string value	Default text to display, when no data is bound
width	measure values	integer (px)	The width of the component

Child Elements

None available.

Actions

hide, show

Events

DoubleClick, Drop, Select

Methods

setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<label name="label1" width="100" bold="true" type="link" fgcolor="#ff0000" default="This is a label"/>
```

Example: UI XML for the Label component

Related Components

None available.

List

Defines a list component that displayes item entries populated by the bound data, which can be selected in different ways. Unlike the ComboBox where only one option can be seleced, the list component makes it possible to select several options.

Usage

The list component is useful for dealing with many entires and offers a simple but flexible way of render it to end-users. The component is controlled by attributes and child elements. First is the header attribute which displayes a header in the top of the component.

The mode attribute is to control the behavior of the component. The default mode value is the multiple, which allows for multiple selections and where the lists is rendered as a normal list. The mutex mode allows for only one single selection. Finally the columns mode will render the list as two separate lists, were selections are made by moving entries from left to right.

Using the attributes bgClass, headerClass and itemClass we can apply CSS classes to the different parts of the list component.

The list has two child elements, the header and the item. Here the header creates an unselectable text, and the item creates selectable text options. Both use the display and match attributes to set an Expression to determine what data should be selectable and what should be displayed.

Using setSelection you can set define the selection that will be used in the list.

Details

Component Type: Structure

Component Properties: High Level Element, Data Driven Component

Parent Elements: Panel

Child Elements: header, item

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px)	The height of the component
width	measure values	integer (px)	The width of the component

Optional Attributes

Name	Type	Values	Description
bgClass	string	Any string value	Name of the CSS class that controls the background
draggable	boolean	**false**, true	If list items should be draggable
header	string	Any string value	If set a header label will be shown
headerClass	string	Any string value	Name of the CSS class that controls the header
itemClass	string	Any string value	Name of the CSS class that controls the items
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
mode	enumeration	columns, **multiple**, mutex	Sets the behavior mode of the component
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
tabindex	numerical	[0...n]	Tab stop index of the component

Child Elements

header

The header element defines headers in the list component. The match attribute in the item element defines what elements in the data XML should be considered header items, and the display attribute defines the look of the header using XPath expressions.

Name	Type	Values	Description
display	string	**@name**, Expression	Required. Relative to the match attribute. The text displayed
match	string	Expression	Required. What data is to be selected.
class	string	Any string value	Optional. CSS class for section

item

The item element defines items in the list component. The match attribute in the item element defines what elements in the data XML that should be bound, and the display attribute defines what will be displayed of the list item using XPath expressions.

More information

Actions

hide, show

Events

ContextMenu, DoubleClick, Drop, Empty, MoveItem, Release, Select

Methods

resetSelection

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work.

```
<panel name="mainPanel" type="row" look="plate" padding="5">
<list name="list1" header="Components" mode="columns" width="auto" height="210">
  <header match="components" display="concat('Components',':')"/>
  <item match="component" display="@name"/>
</list>
</panel>
```

Example: UI XML for the List component

```
<components>
<component tag="Component" name="Accordion" group="UI XML">
  <documentation>
    <description>Accordion description</description>
  </documentation>
</component>
<component tag="Component" name="Browser" group="UI XML">
  <documentation>
    <description>Browser  description</description>
  </documentation>
</component>
<component tag="Component" name="Button"  group="UI XML">
  <documentation>
    <description>Button description</description>
  </documentation>
</component>
</components>
```

Example: Data XML for the List component

Related Components

None available.

ListView

Defines a very flexible component that is driven by an XSLT document as well as data, which means in order for the component to work you need to have a external XSLT document and you need to have an XML data document bound to the component for it to display anything.

By binding data to the component and then transforming that data using XSL transformations you can make data lists that are clickable, selectable and draggable.

Usage

It is a very flexible component when it comes to rendering content since it allows you to create pretty much you own component by designing it and determining what events will be made when clicking on certain elements.

In the listview the developer is able to use HTML, CSS and JavaScript which allows for a great looking and dynamic component.

There are several attributes that are used to control the look and behavior of the listview. First is the deselect attribute, which makes it possible to deselect a selected value inside the list. The focus attribute makes to listview scroll to the selected value. Also note that the height attribute which defines the height of the component will when set to "*" renders the listview depending on the height of the rendered contents. If you set it to "auto" it will take the remaining space.

The normal listview will escape the < and > characters, these can however be obtained by setting the listview mode attribute to xml, so that they are not escaped.

Setting the dimension attribute to true will allow you to extract from height and width from the listview component. They are available as .getParam('height') and .getParam('width').

In the XSLT document, there needs to be an enclosing div element, which must have the selectionBase attribute. This is a required attribute which is needed for unique selections. It is this value together with the underscore that form the prefix used in id's of all clickable items. There are also two additional attributes called selectionClass and hoverClass which are used change the CSS class of the selected item and change the CSS class of the hovered item. See the example at the bottom of this page to understand the context of these attributes.

The item level elements, that are to be clickable, have three required attributes. The id attribute which is to be unique and consists of the prefix we mentioned earlier and the position() function. They are constructed by selectionBase_[n] where n is a item unique number, such as its position in the XML document. The position attribute holds the attribute of the item, and is also generated by the position() function. Finally the xpath attribute contains the absolute xpath of the item.

The special case attribute is the triggerEvent, which is not required. It does however offer nice possibilities, since it allows for creating and receiving listview specific events. In the example below we create a triggerEvent called cName which in later can listen for in the Process XML.

Listviews support multiple selections and lasso features. The multiple selection feature only works within the same data document. This is possible by setting the lasso attribute to true. You can also support multiple selections without the lasso feature, by setting the multiple attribute to true. Observe that both of these are false by default. Multiple selection is also possible by using CTRL+key or Mouse click for selection.

Using setSelection you can set define the selection that will be used in the listview.

Lazy Loading of Images

The listview component allows for lazy loading of images. Meaning that they will load after the the application has been rendered making it faster and more responsize on start up. To be able to use this feature rename your **img** HTML elements to **ximage** elements and then set a **mode** attribute with the value "async".The image created should also be cross-browser capable, meaning that the framework will make the image appear correctly in all supported browsers.

JavaScripting

The listview can contain a single script tag in the rendered code from which JavaScript can be included and executed.

Details

Component type: Structure

Component Properties: High Level Element, XSLT Driven Component, Data Driven Component

Parent Elements: Container, Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px), percent (%), auto, *	The height of the component
width	measure values	integer (px), percent (%)	The width of the component
href	string	URL	Reference to XSLT document used for rendering the listview

Optional Attributes

Name	Type	Values	Description
clickeditable	boolean	**false**, true	Enable ability to rename items by clicking on them
deselect	boolean	**true**, false	If deselection should be possible
dimension	boolean	**false**, true	Make it possible to extract height and width from listview
focus	string	**false**, true, x, xy, view, center, bottom, top, scroll	If/How the listview should be scrolled to the selection
label	string	Any string value	The label text
lasso	boolean	**false**, true	Allow lasso selection of multiple items
lstyle	string	CSS Syntax	Overrides the default label style using CSS
mode	enumeration	**default**, xml	Decides what type of mode to use
moving	string	**false**, true, horizontal, vertical, inside	Defines possibilities to drag and position items inside the listview
multiple	boolean	**false**, true	Allow multiple selections

partial-updates	boolean	true, **false**	If partial updates to the rendered content should be made instead of full re-rendering of content
reload	boolean	true, **false**	If F5 reloading should be turned off
rightselect	enumeration	**true**, false, noevent	If rightclick selection should be enabled or enabled without returning av event
scroll	enumeration	**auto**, scroll, hidden, false	Controls the scroll behavior
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
wait	string	Any string value	Text to display while loading data to component

Child Elements

None available.

XSLT

The XSLT must generate HTML of a certain format. It needs a container div with the following attributes:

Attributes (Root Level)

Attribute	Type	Description
selectionBase	string	Required. Text that (together with underscore) will prefix the ID of all div elements that are items
selectionClass	string	Optional. Class applied to selected list items
hoverClass	string	Optional. Class applied when hovering list items

The HTML elements (divs) that represent list items also need certain attributes:

Attributes (Item Level)

Attribute	Type	Description
id	string	Required. ID of form [selectionBase]_[n], for example if selectionBase attribute in container is "myitem", then IDs are generated by "myitem_{position()}"
position	number (positive integer)	Required. Position of element, use {position()}
xpath	XPath expression	Required. The absolute xpath of the node. For value see the example below
draggable	boolean	Optional. Default is false. if item should be draggable
dragover	boolean	Optional. Default is false. If target item space should be visible while hovering with a draggable item
dragvisible	boolean	Optional. Default is false. If items dragged should be visible (60% opacity).
title	string	Optional. Tool-tip for element
unselectable	boolean	Optional. Defines if the item should be unselectable.

TriggerEvent is an attribute that can be applied to any HTML element inside the item level element. That means it can be applied to img, span and div tags as long as they are located inside the item element of the listview.

The triggerEvent attribute looks like any attribute but will make an special event, that you can catch in the process file. The event made by the triggerEvent is whatever the string value in the attribute is, meaning that it is whatever you want it to be.

Attributes (Sub Item Level)

Attribute	Type	Description
triggerEvent	string	Optional. A string value that can be caught like any regular event.

Actions

hide, show

Events

ctrl+a, ContextMenu, Drop, DoubleClick, Edit, EditEnd, Menu, MoveItem, Release, Return, Select, TriggerEvent

Drop Event Coordinates

You can acquire the coordinates to which something has been dropped. The XML node dropped (remember that files are just atom:entry nodes) will be available in the trigger (!) and the coordinates will be available in !%x and !%y repectively.

```
<alias name="triggerData" value="!"/>
<alias name="x" value="!%x"/>
<alias name="y" value="!%y"/>
```

Methods

activate, addExternalDocument, clear, edit, focusSelection, getParam, getSelectedItem, getValue, haveSelection, resetSelection, setParam, setView, updateRedraw

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work and that it consists of an UI XML part, a XSLT file and data.

Example (UI XML)

```
<listview name="listviewExample" scroll="auto" reload="false" width="auto" height="auto" href="home://Documents/listview.xsl"/>
```

Example: UI XML for the ListView component

Example (XSLT)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xios="urn:x-i-o-s:util" version="1.0">

<xsl:template match="components">
  <div selectionBase="componentOption" selectionClass="selectedItem" hoverClass="hoverItem">
    <xsl:apply-templates select="component"/>
  </div>
</xsl:template>

<xsl:template match="component">
  <div xpath="{xios:absolute(.)}" id="componentOption_{position()}" position="{position()}" draggable="false" class="regularItem">
    <div triggerEvent="cName"><xsl:value-of select="@name"/></div>
    <div triggerEvent="cGroup"><xsl:value-of select="@group"/></div>
  </div>
</xsl:template>

</xsl:stylesheet>
```

Example: XSLT for the ListView component

The above XSLT example will match the root (components) of the XML file below, and create a container div which is required for listviews. This div will has the selectionBase attribute which we will set to componentOption. It can be set to anything, as long as it is unique for your application.

We proceed to apply templates on the component element. There we create a variable named xpath which will contain the absolute xpath of every component element.

Next we create a div element with some special attributes. These are xpath, id, position and draggable. The xpath attribute will contain the xpath variable and thus the absolute xpath of the book element.

The id attribute will contain a combination of the value of selectionBase and the position of the element; thus creating a unique name. The position attribute will contain the position of the element. And finally the draggable attribute will if set to true allow the item to be draggable across the listivew.

Inside the div we will output the value of the name attribute of the component element using xsl:value-of.

Example (XML)

```
<?xml version="1.0"?>
<components>
<component name="Accordion" tag="Component" group="UI XML">
  <documentation>
    <description>Accordion description</description>
  </documentation>
</component>
<component name="Browser" tag="Component" group="UI XML">
  <documentation>
    <description>Browser  description</description>
  </documentation>
</component>
<component name="Button" tag="Component" group="UI XML">
  <documentation>
    <description>Button description</description>
  </documentation>
</component>
</components>
```

Example: XML data file for the XSLT example above

Related Components

Render

Map

Defines a map component utlizing Google Maps.

Usage

The component creates an instance of Google Maps. An additional built in functionality is that you are able to bind XML data to the component which are then rendered as markers on the map according to the coordinates in the data document. By modifying that data document you can effectively control what is displayed on the map, clear the markers and add new ones.

There are several attributes that allows you to customize the behavior and appearance of the component. You can override the default height and width values of the component by adding their respective attributes. The **zoomlevel** attribute defines the zoom level that is used by the map. An appropriate value for this attribute might be about 5 or 6 depending on what you want the end-user to see. You can also define a start position for the map using the **mapcenter** attribute. Observe that the value for this attribute should be a coordinate for example: "58.416139, 15.626392"

The **iconMarker** attribute defines the icon that will be displayed where the marker is loaded. By using the**iconSelect** attribute you can define the icon to use once the end-user starts to drag the marker. It might perhaps be a similar icon to indicate that the marker is being dragged. The **iconTarget** attribute defines which icon should be displayed once a search result has been marked.

Details

Component Type: System

Component Properties: -

Parent Elements: Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
controls	boolean	**true**, false	Defines if controls should be visible
height	measure values	integer (px), percent (%)	The height of the component which will overrule the default value
iconMarker	string	URL	URL to the icon displayed when data is bound to the map
iconSelect	string	URL	URL to the icon displayed when marker is selected
iconSize	numerical	integer (px)	The size of marker icons on the map.
mapcenter	string	decimal value; decimal value	Coordinates to define map center
maptype	enumeration	**roadmap**, satellite, hybrid, terrain	Defines which maptype to use when rendering initial map
style	string	CSS Syntax	Overrides the component's default style
width	measure values	integer (px), percent (%)	The width of the component which will overrule the default value
zoomlevel	numerical	1...[n]	Defines zoom level to be applied on the map

Child Elements

None available.

Actions

None available.

Events

Click, Ready, Move

Methods

getLat, getLng, getMapType, getZoom, searchMap

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<map name="map" width="100%" height="100%" iconMarker="icons/iconMarker.png" iconSelect="icons/iconSelect.png" iconTarget="icons/iconTarget.png"/>
```

Example: UI XML for the Map component

Related Components

None available.

Media

Defines a media component that can play audio and video using "Microsoft Windows Media Player".

Usage

The component can be entirely controlled by methods but has also built in controls which are visible when the **controls** attribute has been set to true. However the component can be completely hidden by setting the **width** and **height** attributes 0px.

Using the **mode** attribute one can easily define the purpose mode of the component. The attribute can be set to either "audio", which is the default state, or to "video".

The **progress** attributes defines if a progressbar should be displayed while playing back the media file. It will appear as a slider attached to the player and can be used to set a new progress position for the file.

When a new songs starts to play the component will fire a "Select" event.

Details

Component Type: System

Component Properties: Child Element

Parent Elements: Panel

Child Elements: None available

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
controls	boolean	**true**, false	Decides if controls should be visible
height	measure values	integer (px), percent (%)	The height of the component
mode	enumeration	**audio**, video	The playmode of the component
progress	boolean	**true**, false	Decides if a progressbar should be displayed
width	measure values	integer (px), percent (%)	The width of the component

Child Elements

None available.

Actions

None available.

Events

Complete, Loading, MediaEnded, MediaRange, Select, UpdateTime

Methods

adjustVolume, getCurrentPosition, getVolume, next, pause, play, previous, repeat, setMode, setMute, setValue, setVolume, showControls, shuffle, stop

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<media name="mediaControl" controls="false" progress="false" mode="audio" height="0"></media>
```

Example: UI XML for the Media component

Related Components

Menu

Defines a menu plate that contains items, which are defined as child elements, or data items which are control by or Rules.

Usage

The component itself acts as a wrapper (container) for items and it controlled by attributes and can be evoked from either a MenuBar or by a ContextMenu.

By setting the caption attribute to true and setting the text attribute you can create a caption footer. You can also grey out the entire menu by setting the enabled attribute to true, this is only available if the menu is part of another menu. The icon attribute allows you to add an icon to the menu option, which will be displayed to the left of the text.

The child element item, has several style and function attributes which can be used. The attributes name and text are required since they are needed to make a basic menu option. Also here the enabled attribute is available with the same function.

Menu items can also be grouped together using the group attribute, and then determine which menu option will be preselected using the selected attribute. The group behavior is also known as mutex functionality.

Details

Component Type: Placeholder

Component Properties: Child Element, Wrapper Component

Parent Elements: MenuBar, View

Child Elements: DataItem, Divider, Menu Item

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
caption	boolean	true, **false**	If a caption text should be shown at the bottom
enabled	boolean	**false**, true	If the menu should be enabled when part of menu
inlinemenu	string	Any string value	If a menu should be displayed as a titlemenu instead
look	enumeration	**default**, graphite	Graphical look of the menu
style	string	CSS Syntax	Overrides the component's default style
submenu	string	Any string value	Reference to menu component to load menu options
text	string	Any string value	Text to show if caption is set to true
title	string	Any string value	Tool-tip text

Child Elements

Item

Defines static menu items in application menus. Menu items can exhibit radio button behavior and checkbox behavior. To define which item should be selected when menu items are grouped, using the group attribute, you use the select action on the menu item.

More information

Actions

enable, disable, hide, show

Events

Init, Show

Methods

setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<menu name="mFile" caption="false" text="File">
  <item name="mOpenDoc" text="Open" icon="xios/icons/applications/16x16/document_into.png"/>
  <divider name="div1"/>
  <item name="mNew" text="New" icon="xios/icons/applications/16x16/document_new.png"/>
  <divider name="div2"/>
  <item name="mSaveDoc" text="Save" icon="xios/icons/applications/16x16/disk_blue.png"/>
  <item name="mSaveAsDoc" text="Save As..." submenu ="mFileTypes" icon="xios/icons/applications/16x16/save_as.png"/>
</menu>
```

Example: UI XML for the Menu component (with submenus)

Related Components

DataItem, Item, MenuBar

MenuBar

Defines a menu bar that is filled with menu component elements. The menubar component only defines a place and a frame for menu components, it does not define the menu plates or menu items.

Usage

The MenuBar is used to create a regular menubar (one that is located in the top of most applications) and that has expandable menu plates containing one or more menu items which can also be data driven.

The menubar component can also be filled with buttonbox components when the attribute "type" is set to tab.

If the position attribute is set to right or left the type attribute must be set to tab.

Details

Component Type: Navigation

Component Properties: High Level Element

Parent Elements: Panel, View

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
position	enumeration	**top**, bottom, right, left	Position of the menubar

Optional Attributes

Name	Type	Values	Description
compact	boolean	**false**, true	Defines if the menubar should be positioned next to the window handling buttons
label	string	Any string value	The label text
look	enumeration	**default**, graphite	Visual apperance for the menubar
lstyle	string	CSS Syntax	Overrides the default label style using CSS
opacity	numerical	**100**, [0-100]	Percentage of the opacity (visibility)
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, tab	appearance type of menubar

Child Elements

None available.

Actions

hide, show

Events

None available.

Methods

None available.

Syntax and Examples

These examples work on the fly and does not need to be bound to data to work.

```
<menubar name="menuBar" draggable="true" position="top">
  <menu name="mFile" caption="false" text="File">
    <item name="mOpenDoc" text="Open" icon="xios/icons/applications/16x16/document_into.png"/>
    <item name="mNew" text="New" icon="xios/icons/applications/16x16/document_new.png"/>
    <item name="mSaveDoc" text="Save" icon="xios/icons/applications/16x16/disk_blue.png"/>
    <item name="mSaveAsDoc" text="Save As..." icon="xios/icons/applications/16x16/save_as.png"/>
  </menu>
</menubar>
```

Example: UI XML for the MenuBar component

```
<menubar name="menuBar" draggable="true" type="tab" position="top">
  <group name="fileGroup" mode="mutex">
    <buttonbox name="bbNew" icon="xios/icons/applications/24x24/document_new.png" text="New" title="New"/>
    <buttonbox name="bbDelete" icon="xios/icons/applications/24x24/document_delete.png" text="Delete" title="Delete"/>
  </group>
  <buttonbox name="bbEdit" icon="xios/icons/applications/24x24/document_edit.png" text="Edit" title="Edit" mode="toggle"/>
</menubar>
```

Example: UI XML for the MenuBar component using buttonboxes with mutex and mode toggle

Related Components

ButtonBox, Item, Menu

Panel

Defines a panel which are infrastructure building blocks in applications, used for arranging other components and creating a desirable layout.

Usage

Panels are entirely controlled by their attributes. You are offered several ways of styling your panel using the look attribute which is used to define the graphical look of the panel. There are many different types of values for this attribute. The default value will render a blue background. The graphite look is only available if the mode attribute has been set to expander.

By using the style attribute you can completely restyle your panels. There are however several other attributes that are used to style the panel. The bgcolor attribute is used to set the color of the title of panels that have one. While border sets the border width, and the opacity attribute sets the visibility in percent. The latter means that you can make your panels transparent.

Several attributes also control the position of the components, such as the align attribute which sets the horizontal alignment of all components within it and the valign which sets the vertical alignment.

The type attribute controls the type of layout the panel should use. Several options are available. The flow attribute will allow the components inside the panel to line up next to each other if space if available and then start at the next row if the space is to short. When the attribute is set to column, every component inside the panel will get its own column. While when type is set to row, every component will get their own row.

The fill attribute is a placeholder attribute that, if set to true will fill the remaining space in the panel with a invisible panel. Hide is a very useful attribute that completely hides the panel and all components within when set to true.

When you are working with the TabStrip component or the Accordion the attribute selected is used to define which panel is preselected.

The mode attribute defines what visual mode the panel should use. The default value will create a regular panel, while the value closable will have a header and a button for closing the panel. The button behavior is built in so there is no need to use Process XML to do so. The frame and heading modes will create panels with prominent headers. Frame mode has a bigger title while the heading mode has a smaller title. Setting this attribute to "glossy" for a frame looking panel and "expander" will make it possible to expand and collapse the panel only displaying the title bar. Other panel frame modes includes aqua, aquabar, round and side.

Observe that if you set a title on a panel it is not a tool-tip, it is a header. If a mode is not chosen and a title attribute is set the panel will render a border that will make the panel appear to be inset (sunk in to the background).

Xlink can be used to show entire view files inside Panels, this is done by using the xlink attributes xlink:actuate, xlink:href and xlink:type together with the xlink namespace.

As an optimization tip, try to set all your panels to fixed sizes, since it redrawing percent is much slowing.

Details

Component Type: Placeholder

Component Properties: High Level Element, Wrapper Component

Parent Elements: Panel, View

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%), auto	The width of the component
height	measure values	integer (px), percent (%), auto	The height of the component

Optional Attributes

Name	Type	Values	Description
align	enumeration	**left**, right, center	Sets alignment of child components within the panel
bgcolor	color	[#rrggbb], transparent	Specify the background color for titled panels
border	enumeration	**default**, left, right	Defines if a border should be used
closeable	boolean	**false**, true	Defines if the panel should be closable when part of a tabstrip
expanded	boolean	**true**, false	Default state of panel when mode is "expander"
fill	boolean	**true**, false	Decides if an invisible element should fill the remaining space in the panel
hide	boolean	true, **false**	If the panel and its contents should be hidden when loaded
icon	string	Any string value	Path to icon when used inside other components like tabstrip and accordion
label	string	Any string value	The label text
look	enumeration	**transparent**, cell, white, alert, dark, black, title, light, window, window2, plategrad, plate, back, gradient, gradient2, gradient3, gradientblue, gradientgray, inset, insetwhite, important, outset, graphite, graphiteoutset, green, pink, purple	Graphical look of the panel
lstyle	string	CSS Syntax	Overrides the default label style using CSS
menu	boolean	**false**, true	Renders a small navigation arrow in the title that when clicked fires the "Menu" event
mode	enumeration	**default**, aqua, aquabar, closeable, expander, frame, glossy, header, round, side	Defines the visual mode of the panel
opacity	numerical	**100**, [0...100]	Specifies the percentage of the transparency
padding	numerical	[0...n]	The amount of padding that should be used
position	enumeration	**default**, bottomright	Define positioning when type is flow
resize	boolean	true, **false**	Set to true if panel should be resizable
scroll	enumeration	**true**,false, auto	Controls the scroll behavior
selected	boolean	true, **false**	If the panel is should be selected
style	string	CSS Syntax	Overrides the component's default style
tabcolor	string	colorname, [#rrggbb]	Color of the tab of the panel when using a tabstrip
title	string	Any string value	The heading of the panel
type	enumeration	**flow**, column, row	The layout to use for content inside panel
valign	enumeration	**top**,bottom, center	Vertical alignment of child components
xlink:actuate	enumeration	onRequest	When the view should be loaded
xlink:href	string	URL	Path to the view file to use
xlink:type	enumeration	simple	Type of link to use

Child Elements

None available.

Actions

hide, show

Events

Close, Drop, Init, Menu

Methods

getValue, maximize, restore, update, setTitle

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```xml
<panel name="panelContainer" type="row">
  <panel name="panel0" height="50" look="white">
    <label name="lab0" default="Normal panel without title attribute"/>
  </panel>
  <panel name="panel1" height="50" look="white" title="Regular Panel">
    <label name="lab1" default="Normal panel with title attribute"/>
  </panel>
  <panel name="panel2" height="50" look="white" mode="closeable" title="A closable Panel">
    <label name="lab2" default="Closable mode panel"/>
  </panel>
  <panel name="panel3" height="50" look="white" mode="frame" title="A panel with a frame">
    <label name="lab3" default="Frame mode panel"/>
  </panel>
  <panel name="panel4" height="50" look="white" mode="header" title="When using the header mode">
    <label name="lab4" default="Header mode panel"/>
  </panel>
</panel>
```

Example: UI XML for the Panel component (All modes)

Related Components

None available.

Plate

Defines a plate component that serves as a container for item elements. Plate is often associated with the ComboBox component.

Usage

The reason that plate is a separate component is that several comboboxes can use the same plate, like in the example at the bottom of this page. If separate components are utilizing the same plate they should stil display different selections.

The Plate is used to display items as options inside a ComboBox. This is accomplished by using rules to match data using Expressions and output item elements, or by manually setting each item option.

Using setSelection you can set define the selection that will be used in the plate.

Details

Component Type: Placeholder

Component Properties: Child Element, Wrapper Component

Parent Elements: ComboBox

Child Elements: None available.

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px)	The height of the component

Optional Attributes

Name	Type	Values	Description
scroll	boolean	**true**, false	If scroll should be visible
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

item

Item elements are single options available inside the plate. Each item element represents a single option. The element itself contains information about what the option should display.

More information

Actions

None available.

Events

None available.

Methods

getText, getValue

Syntax and Examples

The example works on the fly and does not need to be bound to data to work. In order for the second example to work it needs the data provided below to be bound to the component.

```
<combobox name="cb_Components" width="150" height="14" editable="false">
<plate name="plate_Components" height="120" scroll="true">
  <item value="A">A</item>
  <item value="B">B</item>
  <item value="C">C</item>
</plate>
</combobox>
```

Example: UI XML for the Plate component used inside a ComboBox component

```
<plate name="plate_Components" height="120" scroll="true">
<rule match="component">
  <item value="{@name}">{@name}</item>
</rule>
</plate>
<combobox name="cb_Components1" width="150" height="14" editable="false" subplate="plate_Components"/>
<combobox name="cb_Components2" width="150" height="14" editable="false" subplate="plate_Components"/>

```

Example: UI XML for the Plate component used outside a ComboBox component where both comboboxes are using the same plate.

```
<components>
<component name="Accordion" tag="Component" group="UI XML">
  <documentation>
```

```
    <description>Accordion description</description>
  </documentation>
</component>
<component name="Browser" tag="Component" group="UI XML">
  <documentation>
    <description>Browser  description</description>
  </documentation>
</component>
<component name="Button" tag="Component" group="UI XML">
  <documentation>
    <description>Button description</description>
  </documentation>
</component>
</components>
```

Example: Data XML for example

Related Components

ComboBox

Radio

Defines a radio button, in the form of a clickable circle with an optional text label.

Usage

Several radio buttons can be combined using the group component. Together they will exhibit what is known as radio buttion behavior (mutex). This will allow only one radio button within the group to be selectable at any given time.

When a radio component is evaluated using expressions it will return the value attribute (if checked) or an empty string if unchecked.

By using the postLabel or preLabel attributes you can set a label in front off or behind the radio component. The look attribute can be used to control the visual appearance of the labels.

The value attribute contains the value that the radio component will evaluate to. When the component is intended to be preselected the selected attribute is set to true.

Details

Component Type: Selector

Component Properties: High Level Element

Parent Elements: Panel

Child Elements: None available

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

Optional Attributes

Name	Type	Values	Description
hide	boolean	**false**, true	Defines if the component should be hidden upon start up
label	string	Any string value	The label text
look	enumeration	**default**, bold, boldwhite, white	Graphical look of the label
lstyle	string	CSS Syntax	Overrides the default label style using CSS
postLabel	string	Any string value	Text label placed after the radio button of the radio component
preLabel	string	Any string value	Text label placed in front of the radio button of the radio component
selected	boolean	true, **false**	Set to true indicates that the radio component is selected
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
value	string	Any string value	Value represented by the radio component

Child Elements

None available.

Actions

hide, show

Events

Release, Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<group name="group1">
  <radio name="radio1" postLabel="One" value="1"/>
  <radio name="radio2" postLabel="Two" value="2"/>
  <radio name="radio3" postLabel="Three" value="3"/>
</group>
```

Example: UI XML for the Radio component

Related Components

ButtonBox, Menu

Render

The Render component is used to display data using an XSLT file and requires that you bind data to the component. It offers a fast loading XSLT transformation component and a flexible solutions for application design.

Usage

While having the full power of XSLT support the Render lacks much of the functionality that is available in the ListView component. However since it is slimmer it loads much faster than the ListView.

The component also offers special handling of links, both external and internal (using xpath). This is possible by adding an xpath attribute to internal links pointing to the desired data selection. For external links you use the a-element, just in HTML and you also provide a href-attribute for the desired webpage and a with a target-attribute determining how the link should be opened (normally _blank for opening in new window).

The Render component has limited support for triggerEvents. While you can create you own customized event names, they will not contain the selection data like in the ListView component.

Details

Component Type: Structure

Component Properties: High Level Element, XSLT Driven Component, Data Driven Component

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%), auto	The width of the component
height	measure values	integer (px), percent (%), auto	The height of the component
href	string	URL	Reference to XSLT file used for rendering the content

Optional Attributes

Name	Type	Values	Description
color	string/numerical	Color definition or "contrast"	The text style of the top element of the result will be set to this value. If the value is "contrast" the color will be black or white, whichever contrasts the most to the background.
entities	boolean	**false**, true	Determines if < and > should not be escaped
fade	boolean	**false**, true	Defines if a fade effect should be used between transformations
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
scroll	enumeration	**default**, visible, scroll, false	Determine scroll behavior
selectable	boolean	**false**, true	Defines if all the text inside the component should be selected
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

None available.

XSLT

Note below that you can use the triggerEvent attribute standalone but in order to use the triggerComponent attribute which will allow you to trigger a specific event for another component you need the triggerEvent attribute to define the name of that component.

The following attributes are used inside the XSLT.

Optional Attributes

Name	Type	Values	Description
editableField	boolean	**false**, true	Will make the element contents selectable
triggerComponent	string	qid (#view#component)	The component that the triggerEvent will be triggered for

triggerEvent	string	Any string value	A string value that can be caught like any regular event
xpath	string	XPath Expression	Used for internal links

Actions

hide, show

Events

DoubleClick, Select, TriggerEvent

Methods

setParam, setView

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work and that it consists of an UI XML part, a XSLT file and data.

```
<render name="renderExample" scroll="false" width="auto" height="auto" href="home://Documents/render.xsl"/>
```

Example: UI XML for the Render component

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="components">
  <xsl:apply-templates select="//link"/>
</xsl:template>

<xsl:template match="link">
  <div>
    <xsl:choose>
      <xsl:when test="@type = 'external'">
        <a href="{@href}" target="_blank"><xsl:value-of select="text()"/></a>
      </xsl:when>
      <xsl:when test="@type = 'internal'">
        <span xpath="{@xpath}"><xsl:value-of select="text()"/></span>
      </xsl:when>
    </xsl:choose>
  </div>
</xsl:template>
</xsl:stylesheet>
```

Example: XSLT for the Render component

```
<components>
<component name="Accordion" tag="Component" group="UI XML">
  <documentation>
    <description>Accordion description.  <link xpath="components[1]/component[@name='Button']">Link to Button</link></description>
  </documentation>
</component>
<component name="Browser" tag="Component" group="UI XML">
  <documentation>
    <description>Browser description. <link xpath="components[1]/component[@name='Accordion']">Link to accordion</link></description>
  </documentation>
</component>
<component name="Button" tag="Component" group="UI XML">
  <documentation>
    <description>Button description. <link type="external" href="http://xios3.com" target="_blank">XIOS3.com</link></description>
  </documentation>
</component>
</components>
```

Example: Data XML for the Render component (link example)

Related Components

ListView

RichText

Defines a Rich Text Editor that makes it possible for end-users to style and format text. The component creates P- and SPAN-elements and then applies CSS to them, this is however not visible to the end-user instead it is presented as formatted text. The editors looks like a TextArea.

Usage

This component is used to format text, it is however in need of external controls to be also to do so, such as ButtonBoxes in MenuBar.

The Rich Text Component does accept that you bind data and modify it.

The **wrap** attribute is used to define the wrapping of the component. Wrapping can be set to "off", "soft" or "hard" where "hard" breaks words in order to wrap the content. Where as the "soft" will only apply normal word wrap.

Styling of the text is made possible by using the modStyle method and listening to the events made by the editor, such as the alignCenterSelect and alignCenterRelease and then applying or removing style depending on the event.

The **DoubleClick** event is only thrown when the editor is locked for editing.

Details

Component Type: System

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component
height	measure values	integer (px), percent (%)	The height of the component

Optional Attributes

Name	Type	Values	Description
autoSpellcheck	boolean	**false** , true	Defines if spellcheck should be started automatically
focus	boolean	**true**, false	Defines if the component should be focused when initiated
editable	boolean	**true**, false	Decides if the component should be editable
iframe	boolean	true, **false**	Defines if the richtext should be loaded in an iframe
label	string	Any string value	The label text
look	enumeration	**default**, graphite	Defines the visual appearence of the component
lstyle	string	CSS Syntax	Overrides the default label style using CSS
scroll	enumeration	**auto**, scroll, false, true	Controls the scroll behavior
style	string	CSS Syntax	Overrides the component's default style
tabindex	numerical	1...[n]	Defines the tab order
title	string	Any string value	Tool-tip text
wrap	enumeration	hard, **soft**, off	Word wrap mode for richtext component

Child Elements

None available.

Actions

hide, show

Events

AlignCenterSelect, AlignCenterRelease, AlignLeftSelect, AlignLeftRelease, AlignRightSelect, AlignRightRelease, AlignJustifiedSelect, AlignJustifiedRelease, BGColorSelect, BGColorRelease, Blur, BoldSelect, BoldRelease, BulletListRelease, BulletListSelect, Change, ContextMenu, DecorationSelect, DecorationRelease, DoubleClick, Drop, FGColorSelect, FGColorRelease, Focus, FontFamilySelect, FontFamilyRelease, FontSizeSelect, FontSizeRelease, ItalicSelect, ItalicRelease, Key, NumberListSelect, Release

Methods

clear, disableAutoSpellcheck, enableAutoSpellcheck, executeCommand, getLineBottomMargin, getLineTopMargin, getSelectedText, getValue, insertHyperlink, insertImage, insertText, modStyle, print, removeList, setEditableMode, setLineMargin, setLineIndent, wordWrap

Syntax and Examples

This example works on the fly and does not need to be bound to data to work. However note that this component is controlled by methods (and events), so for full functionality they must be utilized.

```
<richtext name="richTextEditor" height="auto" width="auto" wrap="soft"/>
```

Example: UI XML for the RichText component

Related Components

TextArea

Shell

Defines a text-based command line interface. The Shell is used for text based communication with the XIOS/3 system.

Usage

The Shell component is an application component that can be integrated into any application. The component can execute the same commands as the Shell application on your desktop.

Details

Component Type: System

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: banner, look, macro, group, command

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component

height	measure values	integer (px), percent (%), auto, fill	The height of the component
width	measure values	integer (px), percent (%), auto, fill	The width of the component

Optional Attributes

Name	Type	Values	Description
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
scroll	enumeration	auto, scroll, visible, false	Controls the scroll behavior
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text

Child Elements

banner

If present the text of the banner element will be printed when the shell is opened.

Name	Type	Values	Description
xios	boolean	true, **false**	If true a generated banner is printed out, using the title, version, and copyright of the system document.

look

Defines a look for the shell. A look can be selected from the look command, from the shell look attribute, or from the theme system. The available looks can be listed using the look command with no argument.

Name	Type	Values	Description
bgColor	color	#rrbbgg	The color of the shell background.
confirmation	color	#rrbbgg	The color used to display confirmation messages.
dirColor	color	#rrbbgg	The color used to display directories/folders.
divider	color	#rrbbgg	The color of dividers.
error	color	#rrbbgg	The color used for error messaged.
fileColor	color	#rrbbgg	The color used to display files.
font	string	LucidaConsole, Fixedsys, CourierNew, AndaleMono, Consolas, Terminal, Courier, Monaco, NimbusMonoL, DejaVuSansMono, BitstreamVeraSansMono, Courier10Pitch, OCRAExtended, Droid	The font to use for the shell text.
fontColor	color	#rrbbgg	The color of the shell default font.
fontSize	enumeration	tiny, smallest, smaller, medium, larger, largest, huge	The size of the shell text.
formAnswer	color	#rrbbgg	The color used for the answer or content of a form.
formText	color	#rrbbgg	The color used for the question or title for a form.
info	color	#rrbbgg	The color used for information text.
name	string	unique name	The name of the shell look.
progress	color	#rrbbgg	The color used for wait messages.

macro

Defines a command as a macro, i.e. a different command line. The content of the element is the command line that the macro efines. The dollar sign acts as a variable for a macro argument, e.g. a macro named "delete" could have the definition "rm $" to remove the file given after the delete command.

Name	Type	Values	Description
description	string	Any string value	A short description of the macro, displayed from the macro command.
name	string	uniqe name	The name of the macro.

group

Defines a command group and places all commands defined by the child command elements into that group. The group is used in the grouping in the help command.

Name	Type	Values	Description
name	string	uniqe name	The name of the group.

command

Defines a command. The content is the description of the command. A command can either be defined internally in the Shell component, or it can be defined in process code where every non-internal command will trigger a Command_name event from the shell component.

Name	Type	Values	Description
internal	enumeration	any of the internal commands	Select a command from the internal list of defined commands.
name	string	uniqe name	The name of the command.

Actions

hide, show

Events

Every non-internal command will trigger a "Command_name" event, where the "name" is the defined name of the command. The element in the trigger will be a command node with a "name" and "path" attribute with the name of the command and the current path of the shell. The parsed options will be in the "option" elements and the left over arguments in "argument" elements. The unparsed command line is in a "raw" element. Finally, if this is part of chained commands, the previous result is in a "result" element.

If there are no registered commands for a specific input, the "Command" event is triggered.

ChangedBackground, ChangedFont, ChangedLook, Command, ContextMenu, Close, Drop, Fullscreen, Menu, Tab

Methods

getBackground, getFont, getLook, setLook

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<shell name="myShell" width="100" height="100" scroll="auto"/>
```

Example: UI XML for the Shell component

Related Components

None available.

Slider

Defines a component that will allow the user to choose from a limited about of values by dragging a slider across the a bar.

Usage

All values are predefined and contained within a plate. When the user drags the slider it will change values while also outputting the value above the slider.

The component has an optional label attribute which will output a text label above the component when set.

Details

Component Type: Navigation

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: plate

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px)	The width of the component
height	measure values	integer (px)	The height of the component

Optional Attributes

Name	Type	Values	Description
label	string	Any string value	The label text
look	enumeration	**default**, graphite	Defines the visual appearence of the slider and the plate
max	numerical	1 or higher	The maximal possible value. Must be higher than the min attribute
min	numerical	0 or higher	Tha minimal possible value. Must be lower than the max attribute.
lstyle	string	CSS Syntax	Overrides the default label style using CSS
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, media, toolbar	Visual mode of the slider

Child Elements

None available.

Actions

enable, disable, hide, show

Events

Select

Methods

getValue, setValue

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<slider name="sliderExample" label="Arc Size" width="120" look="graphite" type="toolbar">
<plate name="arcSizePlate">
  <item value="0.0" selected="true">0%</item>
  <item value="0.05">10%</item>
  <item value="0.10">20%</item>
  <item value="0.15">30%</item>
  <item value="0.20">40%</item>
  <item value="0.25">50%</item>
  <item value="0.30">60%</item>
  <item value="0.35">70%</item>
  <item value="0.40">80%</item>
  <item value="0.45">90%</item>
  <item value="0.50">100%</item>
</plate>
</slider>
```

Example: UI XML for the Slider component

Related Components

ComboBox

TabStrip

The TabStrip Component handles and shows tabs.

Usage

Each tab is actually defined by a Panel component which is located inside the TabStip Component element. The component uses mutex behavior

TabStrips offer limited options for styling and layout, however there are minor customizations that you can use. First by using the position attribute you can position the tabstip to either the top or bottom. The tabs attribute can be use to hide the tabs to only display the preselected panel.

Observe that some attribute used for controlling the tabStrip are set directly on Panels; these include the selected and icon attributes.

Using the show and hide actions on panels that make up the tabstrip will also emulate that behavior in the tabstrip by changing that is selected.

Using setSelection you can set define the selection that will be used in the tabstrip.

Details

Component Type: Navigation

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component
height	measure values	integer (px), percent (%)	The height of the component

Optional Attributes

Name	Type	Values	Description
font	string	Any string value	Defines the font to use on the text
position	enumeration	**top**, bottom	Specifies the position of the tabstrip
label	string	Any string value	The label text
look	enumeration	**default**, classic, graphite	Defines the visual appearence of the tabstrip
lstyle	string	CSS Syntax	Overrides the default label style using CSS
style	string	CSS Syntax	Overrides the component's default style
tabindex	numerical	1...[n]	Defines the tab order
tabs	boolean	**true**, false	Sets the visibility of the tabstrip
title	string	Any string value	Tool-tip text

Child Elements

None available.

Actions

hide, show

Events

ContextMenu, Release, Select

Events on contained panels

Select

Methods

flashTab, selectTab

Syntax and Examples

This example works on the fly and does not need to be bound to data to work.

```
<tabstrip name="tabstrip1" width="auto" height="auto" position="top" tabs="true">
<panel name="panel1" type="row" selected="true">
  <label name="lab1" default="This a label"/>
</panel>
<panel name="panel2" type="row" icon="xios/icons/applications/16x16/help2.png">
  <label name="lab2" default="This a another label"/>
</panel>
</tabstrip>
```

Example: UI XML for the TabStrip component

Related Components

Panel

TextArea

Defines a Text Area used for displaying and editing text that needs several lines.

Usage

It is a very basic component used to edit large amounts of text, unlike the Input which is used for single line editing.

The component is controlled by attributes, and can be set to two difference modes, using the mode attribute. The default mode will escape any text written, while the xml mode will not.

Using the wrap attribute you can determine how text inside the component should be wrapped. The focus attribute allows for the component to be focused, meaning that the marker will be set in the component. If several components has the this attribute set to true it is applied to the latest rendered component. The enabled attribute makes it possible to disable the component, so that it appears to be greyed out and unusable.

Details

Component Type: System

Component Properties: High Level Element

Parent Elements: Container, Panel

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
width	measure values	integer (px), percent (%)	The width of the component
height	measure values	integer (px), percent (%)	The height of the component

Optional Attributes

Name	Type	Values	Description
default	string	Any string value	The default text value displayed in the component
enabled	boolean	**true**, false	Set to false to disable the component
focus	boolean	true, **false**	If the component should be focused
hide	boolean	**false**, true	Decides of the component should be hidden when initially loaded
hold	boolean	true, **false**	If component should stop listening to data updates
label	string	Any string value	The label text
length	numerical	[1-n]	Defines the maximum character length of the content
look	enumeration	**default**, white	The visual look of the component
lstyle	string	CSS Syntax	Overrides the default label style using CSS
mode	enumeration	xml, **default**	Process text structured as XML or just as plain text
style	string	CSS Syntax	Overrides the component's default style
tabindex	numerical	1...[n]	Defines the tab order
title	string	Any string value	Tool-tip text
wrap	enumeration	**soft**, hard, off	Word wrap mode for textarea

Child Elements

None available.

Actions

enable, disable, hide, show

Events

Changed, ContextMenu, Key, Release, Return, Saved, Select

Methods

activate, clear, execCommand, getSelectionLength, getSelectionValue, getValue, releaseEnter, setFont, takeEnter, wordWrap

Syntax and Examples

These examples work on the fly and do not need to be bound to data to work.

```
<textarea name="Textarea1" height="100" width="100" wrap="soft"/>

<textarea name="Textarea2" height="100" width="100" wrap="hard" enabled="false"/>

<textarea name="Textarea3" height="100" width="100" wrap="off" focus="true"/>

```

Example: UI XML for the TextArea component

Related Components

Input

ToolBar

The toolbar component is a container for ButtonBox components that can be aligned towards any border of the application window or be floating.

Usage

If the toolbar is draggable, using the draggable attribute, then a docked toolbar can be made floating by the end-user and vice versa. The Group component can be used to group buttonbox components together and creates separators between different groups in the toolbar that share behavior.

Grouping several ButtonBox components is done by using the group component and its mode attribute. This will make them share behavior.

The opacity attribute controls the visibility of the component and is set to 100 be default, which means that it has 0 transparency. Different visual modes can also be chosen using the type attribute. The default value will display a greyish toolbar white dark will display a darker version of that toolbar. The value zoom will increase the size of the hovered option, simulating zoom behavior, while system will make the toolbar look like the system tray in XIOS/3.

Positioning is also possible; mainly by using the position attribute. It allows you to position your toolbar in four different directions, and also to be floating. When the value floating is used, then the xpos (left) and ypos (top) attribute can be used. These are used to set the absolute position of the toolbar relative to the application window. Observe that the value float for the position attribute cannot be used when placing the toolbar inside a panel.

Details

Component Type: Placeholder

Component Properties: High Level Element, Container Component

Parent Elements: Panel, View

Child Elements: None available.

Can Use Rules: No

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px)	The height of the component
position	enumeration	top, bottom, left, right, float	Set the position of the toolbar
width	measure values	integer (px)	The width of the component

Optional Attributes

Name	Type	Values	Description
draggable	boolean	true, **false**	If the toolbar should be draggable
hide	boolean	**false**, true	Defines if the component should be hidden upon start up
icon	boolean	**false**, true	Defined if space should be reserved for an icon but not to load an icon at startup
label	string	Any string value	The label text
look	enumeration	**default**, liquid	Overrides the default skin styling for the toolbar
lstyle	string	CSS Syntax	Overrides the default label style using CSS
opacity	numerical	**100**, [0...100]	Specifies the percentage of the visibility
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
type	enumeration	**default**, system, zoom, dark	Defines the visual appearance of the toolbar
xpos	numerical	[1...n]	Defines the left position for a floating toolbar
ypos	numerical	[1...n]	Defines the top position for a floating toolbar

Child Elements

None available.

Actions

hide, show

Events

ContextMenu

Methods

None available.

Syntax and Examples

These examples work on the fly and does not need to be bound to data to work.

```
<toolbar name="toolbarExample" position="top" draggable="true">
<group name="fileGrp">
  <buttonbox name="New" icon="xios/icons/applications/24x24/document_new.png"/>
  <buttonbox name="Open" icon="xios/icons/applications/24x24/folder.png"/>
  <buttonbox name="Save" icon="xios/icons/applications/24x24/disk_blue.png"/>
</group>
<group name="editGrp">
  <buttonbox name="Copy" icon="xios/icons/applications/24x24/copy.png"/>
  <buttonbox name="Cut" icon="xios/icons/applications/24x24/cut.png"/>
  <buttonbox name="Paste" icon="xios/icons/applications/24x24/paste.png"/>
</group>
</toolbar>
```

Example: UI XML for the ToolBar component (normal mode)

```
<toolbar name="toolbarExample" position="top" draggable="false" type="dark">
<buttonbox name="Copy" icon="xios/icons/applications/24x24/copy.png" text="Copy" title="Copy"/>
<buttonbox name="Cut" icon="xios/icons/applications/24x24/cut.png" text="Cut" title="Cut" mode="toggle" />
<buttonbox name="Paste" icon="xios/icons/applications/24x24/paste.png" text="Paste" title="Paste"/>
</toolbar>

```

Example: UI XML for the ToolBar component (dark mode)

Related Components

ButtonBox

Tree

Defines a tree view. It is used to display data in a classic tree structure.

Usage

The tree component uses rules to display branches (nodes). The rule elements use the match attribute to select the relative data bound to the component. By using the display attribute you want output either data or text. If you are using data in the display attribute, observe that the attribute is using relative XPath in relation to the rule.

Only one rule condition can be met per data item. In the example below we have bound the entire data document (root), the reason if that the tree must have a root node. Looking at the UI XML we use the first rule to match the XML data root node, components. We choose to display the string Component List instead of displaying data. We also set the icon for the matched element.

Our second rule condition proceeds to match the component node, which there are three of in the data XML. Again we set a icon, but here we use the display attribute on the rule to display the name attribute on the component element inside the data XML. Our third rule is similar to the first rule.

When using Expressions on the component it will return the data of the selected tree item and its children.

Styling of the component is possible, using in the first the border and bgcolor attribute to the border width and the background color. The focusnew attribute will set focus on newly created nodes inside the tree. This is used when the data bound to the tree is updated.

Using setSelection you can set define the selection that will be used in the tree.

Details

Component Type: Structure

Component Properties: Data Driven Component

Parent Elements: Container, Panel

Child Elements: None available

Can Use Rules: Yes

Attributes

Required Attributes

Name	Type	Values	Description
name	string	unique name	The name of the component
height	measure values	integer (px), percent (%)	The height of the component
icon	string	URL	Path to the icon
width	measure values	integer (px), percent (%)	The width of the component

Optional Attributes

Name	Type	Values	Description
border	numerical	[0...n]	Size of border in pixels
bgcolor	color	[#rrggbb]	Decides the color of background
docmatch	string	Expression	Match a node-set from an external document

focusnew	boolean	**true**, false	Decides if the tree should focus on newly created nodes in the tree
label	string	Any string value	The label text
lstyle	string	CSS Syntax	Overrides the default label style using CSS
multiple	boolean	**true**, false	Allow multiple selections
sort	enumeration	**ascending**, descending	Defines in which order the sorting is listed
sort-key	Expression	XPath	Defines by which XPath the tree nodes will be sorted
style	string	CSS Syntax	Overrides the component's default style
title	string	Any string value	Tool-tip text
wait	string	Any string value	Text to display while loading data to component

Child Elements

rule

The rule element is used to match specific data elements and display them as as a tree structure.

Required attributes

Name	Type	Values	Description
display	string	XPath Expression	XPath expression relative to the match attribute
icon	string	URL	Icon to render right of the display text
match	string	XPath Expression	XPath expression relative to the data bound

Optional attributes

Name	Type	Values	Description
draggable	boolean	**true**, false	If rendered branches of tree should be draggable
icon2	string	URL	Icon to display if the tree branch is expanded
open	boolean	**true**, false	Defines if tree branch should be expanded

Actions

activate, hide, show

Events

ContextMenu, Delete, DoubleClick, Drop, Edit, Select

Methods

edit, getNodePath, getCurrentLabel, getDataLocation, getSrcDataLocation, navigateUp, nodeExpanded, setSelectionEx, setSelectionRoot, toggleNode

Syntax and Examples

Observe that this example needs to be bound to the data provided below in order to work.

```xml
<panel name="panelContainer" type="row" padding="4" look="white">
<tree name="treeExample" border="0" bgcolor="#fff" focusnew="false" width="300" height="auto">
  <rule match="components" icon="icon://star" display="'Components List'" open="true"/>
  <rule match="component" icon="icon://piece" display="@name" open="false"/>
  <rule docmatch="xios/config/types.xml#//foldertypes/type" icon="icon://about" display="@name" open="false"/>
</tree>
</panel>
```

Example: UI XML for the Tree component

```xml
<components>
<component name="Accordion" tag="Component" group="UI XML">
  <documentation>
    <description>Accordion description.</description>
  </documentation>
</component>
<component name="Browser" tag="Component" group="UI XML">
  <documentation>
    <description>Browser description.</description>
  </documentation>
</component>
<component name="Button" tag="Component" group="UI XML">
  <documentation>
    <description>Button description.</description>
  </documentation>
</component>
</components>
```

Example: Data XML for the Tree component

Related Components

None available.

Functions

There are several functions available for developers that are very useful.

The functions allow you to manipulate strings, extract information from document paths och also to work with XML documents.

apply-locale

A function that applys language localization to any XML document.

Syntax

```
XML node-set = apply-locale('[XML Document]', '[XPath]')
```

Example

In the example below the alias $localizedDoc will contain an XML documement where the values have been localized according to the language files.

```
<alias name="localizedDoc" value="apply-locale(data/text.xml)"/>
```

In the example below only a specific node will be localized, not the full document.

```
<alias name="localizedDoc" value="apply-locale('data/text.xml', '/node[1]/node[3]')"/>
```

Related Topics

get-label

capitalize

A function that returns the passed string as capitalized.

Syntax

```
string = capitalize([string])
```

Example

In the example below the alias $newString will contain the string "Internet rocks".

```
<alias name="newString" value="capitalize(internet rocks)"/>
```

Related Topics

downcase, upcase

concat

A function that returns a concatenation of its parameters.

Syntax

```
string = concat([string]...)
```

Example

```
<alias name="email" value="concat('lars','@','domain.com')" model="value"/>
```

count

A function that returns the number of elements matching your Xpath.

Syntax

```
integer = count([xpath])
```

Example

```
<alias name="intItemsInFile" value="count($dataDoc#//item)" model="value"/>
```

count-selection

A function that returns the length of a selection.

Syntax

```
integer = count-selection([xpath])
```

Example

```
<alias name="itemLength" value="count-selection($dataDoc#//item)" model="value"/>
```

changes

A function that returns the number of changes that have been made to a document since it was loaded or saved.

Syntax

```
integer = changes([string])
```

Example

If no changes have been made to the document the alias $changesMade will contain the string '0'.

```
<alias name="myDataDoc" value="home://Documents/file.xml"/>
<alias name="changesMade" value="changes($myDataDoc)"/>
```

channelname

A function that returns the name of the communication channel which provides the document.

Syntax

```
string = channelname([string])
```

Example

In the example below the alias $mainChannel will contain the string "xios".

```
<alias name="newSting" value="channelname(home://Documents/Picture/home.png)"/>
```

Related Topics

downcase, upcase

contains

A function that checks if a string contains a specific string.

Syntax

```
boolean = contains('needle','haystack')
```

Example

```
<alias name="containsFalse" value="contains('false','Singing in falsetto.')"/>
```

Related Topics

None available.

document

A function that returns the name of a document without file extensions (eg. .xml) or folderpath.

Syntax

```
string = document([string])
```

Example

In the example below the alias $documentname will contain the string 'Italy Vacation'.

```
<alias name="documentname" value="document(home://Documents/Italy Vacation.xml)"/>
```

Related Topics

None available.

downcase

A function that returns the passed string in lowercase.

Syntax

```
string = downcase([string])
```

Example

In the example below the alias $newString will contain the string 'hello world'.

```
<alias name="newSting" value="downcase(HELLO WORLD)"/>
```

Related Topics

capitalize, upcase

escape-string

A function that escapes strings for inclusion in expressions. The characters backslash, single quote, and double quote are prefixed with a backslash.

Syntax

```
string = escape-string([string])
```

Example

In the example below the alias $newString will contain the string "Internet rocks".

```
<alias name="newString" value="read('store://Documents/escape-string({$untrustedName})')"/>
```

Related Topics

None available.

filename

A function that returns the name of the file from the URL given to it. This includes the file suffix, if any.

Syntax

```
string = filename([string])
```

Example

In the example below the alias $filename will contain the string "home.png".

```
<alias name="filename" value="filename(home://Documents/Photots/home.png)"/>
```

Related Topics

foldername, folderpath

filetype

A function that returns the filetype XML of the passed document.

Syntax

```
XML = filetype([string])
```

Example

```
<alias name="filetypedata" value="filetype(home://Documents/Photots/home.png)"/>
```

Related Topics

None available.

folderexist

A function that returns either true or false depending if a folder exists when passed a folder path.

Syntax

```
boolean = folderexist([string])
```

Example

In the example below the alias $myFolder will contain false as we set a path that does not exist. It can be used in operation decision to check if a folder exists.

```
<alias name="myFolder" value="folderexist(home://fakeFolder/)"/>
```

Related Topics

None available.

folderid

A function that returns the id of a folder when passed a folder path.

Syntax

```
integer = folderid([string])
```

Example

In the example below the alias $folderId will contain a string of the id number of the folder as found in folders.xml. This id is unique for the folder in the entire filesystem.

```
<alias name="folderId" value="folderid(home://Documents/)"/>
```

Related Topics

None available.

foldername

A function that returns the name of the folder in which the document is located. Ex "home://Documents/Pictures/image.png" as a parameter will return "Pictures".

Syntax

```
string = foldername('[string]|[id]', '[int]')
```

The first parameter is a folder path or folder id and is required. The second parameter which is optional defines the folder depth to return from the root. If no value is provided the function will return the name of the folder which contains the document. If the value '0' is provided it will return nothing, although if '2' is provided it might return 'Pictures' (if the file was located at home://Documents/Pictures/)

Example

In the example below the alias $folderLocation will contain the string "Pictures".

```
<alias name="folderLocation" value="foldername('home://Documents/Pictures/home.png')"/>
```

In the example below the alias $folderListing will contain the string 'Pictures'.

```
<alias name="folderListing" value="foldername('home://Documents/Pictures/home.png', '2')"/>
```

Related Topics

folderpath

folderpath

A function that returns the full folder path of the document. Ex "home://Documents/Pictures/image.png" as a parameter will return "home://Documents/Pictures/".

Syntax

```
string = folderpath('[document]', '[int]')
```

The first parameter is a document URL and is required. The second parameter which is optional defines the folder depth to return from the root. If no value is provided the function will return the path of the folder which contains the document. If the value '0' is provided it will return the channel name (ex 'home://'), and if '2' is provided it will return 'home://Documents/Pictures/'

The second parameter also supports negative index (-1 and so on) which will allow the function to return the parent folder, and -2 will return the parent-parentfolder.

Example

In the example below the alias $DocFolderPath will contain the string 'home://Documents/Pictures/'.

```
<alias name="DocFolderPath" value="folderpath('home://Documents/Pictures/home.png')"/>
```

In the example below the alias $ParentFolder will contain the string 'home://Documents/'.

```
<alias name="ParentFolder" value="folderpath('home://Documents/Pictures/home.png','1')"/>
```

Related Topics

foldername

format-bytes

A function that convertes bytes into megabytes (MB), gigabytes (GB) etc.

Input	Output
< 1 KB	Value in bytes
> 1KB and < 1 MB	Value in kilobytes
> 1MB and < 1 GB	Value in megabytes
> 1GB and < 1 TB	Value in gigabytes
> 1TB	Value in terrabytes

Syntax

```
string = format-bytes(integer)
```

Examples

In the example below the alias $size will contain a more readable size of the passed bytesize.

```
<<alias name="byteSize" value="{$atomEntry#atom:link/@length}"/>
<alias name="size" value="format-bytes({$byteSize})"/>
```

Related Topics

None available.

localizedate

A function that returns the passed system date string as localized to the users settings.

Syntax

```
string = localizedate([string])
```

Example

In the example below the alias $localDate will contain the string "2021-04-09 15:10".

```
<alias name="localDate" value="localizedate(2021-04-09T13:10:54Z)"/>
```

Related Topics

formatdate,

formatdate

A function that returns a specified date formatted according to a specificed format. The following formatting symbols are available that can be used together with other characters to create formatting strings. To escape any of these sequences, preceed it with a backslash (\ ) character.

hh	0-prefixed 12-hour hours, i.e. "07"
h	12 hour hours, i.e. "7"
HH	0-prefixed 24-hour hours, i.e. "19"
H	24-hour hours, i.e. "19"
mm	0-prefixed minutes, i.e. "05"
m	Minutes, i.e. "5"
ss	0-prefixed seconds, i.e. "00"
s	Seconds, i.e. "0"
yyyy	Full year, i.e. "2020"
yy	Last two digits of year, i.e. "20"
dddd	Weekday as text, i.e. "Wednesday"
ddd	Three character abbreviation of weekday, i.e. "Wed"
dd	0-prefixed day of month, i.e. "09"
d	Day of month, i.e. "9"
MMMM	Month as text, i.e. "September"
MMM	Three character abbreviation of month, i.e. "Sep"
MM	0-prefixed month number, i.e. "08"

M	Month number, i.e. "8"
tt	"AM" or "PM"
t	"A" or "P", signifying AM or PM
S	"st", "nd", "rd" or "th" to construct month ordinal like "23rd"

There are a few formatting strings available when used as the only character in an argument.

d	M/d/yyyy
D	dddd, MMMM dd, yyyy
F	dddd, MMMM dd, yyyy h:mm:ss tt
m	MMMM dd
r	ddd, dd MMM yyyy HH:mm:ss GMT
s	yyyy-MM-ddTHH:mm:ss
t	h:mm tt
T	h:mm:ss tt
u	yyyy-MM-dd HH:mm:ssZ
y	MMMM, yyyy

Syntax

```
string = formatdate('[string]', '[string]')
string = formatdate('[string]')
```

Example

In the example below the alias $tomorrow will format tomorrow's date into YY-M-d format.

```
<alias name="tomorrow" value="{formatdate('tomorrow', 'yy-M-d')}"/>
```

In the example below the alias $timeVariable will contain the date of three days from now.

```
<alias name="timeVariable" value="{formatdate('t + 3 d', 'yy-M-d')}"/>
```

In the example below the alias $timeVariable will contain the date of next Monday.

```
<alias name="timeVariable" value="{formatdate('next Monday', 'yyyy-M-d')}"/>
```

Some additional examples:

```
<alias name="timeVariable" value="{formatdate('July 8th, 2018, 10:30 PM')}"/>
```

```
<alias name="timeVariable" value="{formatdate(14:55:22)}"/>
```

```
<alias name="timeVariable" value="{formatdate('december 24', 'dddd')}"/>
```

```
<alias name="timeVariable" value="{formatdate('2018-12-27', 'yy/M/d')}"/>
```

```
<alias name="timeVariable" value="{formatdate('last friday', 'yyyy-M-d')}"/>
```

Related Topics

localizedate, adddate, parsedate

get-label

A function that returns the locale (localized language version) value of the passed label.

This function will return the value of the language that the user has defined in his #Settings or in en_US which is th default version.

This function requires the declaration of the l namespace.

Syntax

```
string = get-label([string])
```

Example

```
<alias name="okText" value="{get-label('open')}"/>
```

Related Topics

apply-locale

index-of

A function that returns the index of a specific string.

Syntax

```
boolean = index-of('needle','haystack')
```

Example

```
<alias name="falseInString" value="index-of('false','Singing in falsetto.')"/>
```

Related Topics

None available.

is-alias

A function that returns checks if an alias has been defined and returns either true if set or false it the alias has not been set.

Note: the alias should be written without the leading $.

Syntax

```
boolean = is-alias([string])
```

Example

```
<operation name="decision">
  <when test="is-alias(username)" step="continueWithStartUp"/>
  <otherwise step="getUsername"/>
</operation>
```

Related Topics

None available.

is-open

A function that returns true or false depending if a view is open or not.

Syntax

The parameter passed is the name of the view with the instance number.

```
integer = is-open(name:instance)
```

Example

In the example below the alias $open will contain true if the Document Explorer application is open.

```
<alias name="open" value="is-open(DE:1)"/>
```

Related Topics

is-running

is-running

A function that returns the number of instances running of an application.

Syntax

The parameter passed is the name of the application.

```
integer = is-running(string)
```

Example

In the example below the alias $instances will contain the number of running Document Explorer instances.

```
<alias name="instances" value="is-running(Document Explorer)"/>
```

Related Topics

is-open

local-name

A function that returns the name of the attribute or element without the namespace. For example if the element "exp:item" would return only "item".

Syntax

```
string = local-name(string)
```

Example

```
<operation name="decision">
  <when test="local-name($documentEntry) = 'title'" step="100"/>
  <otherwise step="200"/>
</operation>
```

match

A function that returns the string of the first match. It uses JavaScript Regular Expressions syntax.

Syntax

```
string = match('[string]','[regexp]')
```

Example

In the example below the alias $emailDomain will contain the string '@domain.com'

```
<alias name="emailDomain" value="match('goran@domain.com', '@[a-zA-Z0-9.-]+\.[a-zA-Z]')"/>
```

Related Topics

replace

parsedate

A function that parses a string that represents a date and returns it in the requested format.

Syntax

```
string = parsedate('[string]','[string]')
```

The first parameter is the is input string and should be a date formatted string.

The second paramter is the format string that defined in which format the output shall be provided.

Only the first parameter is required, the second is optional and the function will return the date in the yyyy-MM-ddTHH:mm:ss format unless otherwise stated.

Example

The example below will return todays date in this format: yyyy-MM-ddTHH:mm:ss

```
<alias name="newDate" value="parsedate(now)"/>
```

The example below will return the date in this format: 2007-07-08T22:30:00

```
<alias name="newDate" value="parsedate('July 8th, 2007, 10:30 PM')"/>
```

Some additional examples:

```
<alias name="newDate" value="parsedate('10/15/2019', 'M/d/yyyy')"/>
```

```
<alias name="newDate" value="parsedate('15-Oct-2019', 'd-MMM-yyyy')"/>
```

```
<alias name="newDate" value="parsedate('2019.10.15', 'yyyy.MM.dd')"/>
```

Related Topics

adddate, formatdate

position

A function that returns the position of the element requested by the Xpath parameter.

Syntax

```
integer = position([xpath])
```

Example

In the example below the alias $randomNumber will contain a string with a number between 1 and 100.

```
<operation name="decision">
  <when test="position(#view1#listview10) = 5" step="100"/>
  <otherwise step="200"/>
</operation>
```

random

A function that returns a random number between x and y.

Syntax

```
integer = random('[int]', '[int]')
```

Example

In the example below the alias $randomNumber will contain a string with a number between 1 and 100.

```
<alias name="randomNumber" value="random('1','100')"/>
```

In the example below we alert the random number in a context.

```
<operation name="debug" value="I have {random('1','100')} cars!"/>
```

Related Topics

None available.

read

A function that returns a read-only version of the passed document.

Syntax

```
XML = read('[string]')
```

Example

```
<alias name="docReadOnly" value="read('home://Documents/important.xml')"/>
```

Related Topics

None available.

replace

A function that returns a string where a substring has been replaced. It uses JavaScript Regular Expressions syntax with the global ("g") flag.

Syntax

```
string = replace('[string]', '[string|RegExp]', '[string]')
```

Example

In the example below the alias $replacedString will contain the string '12'.

```
<alias name="replacedString" value="replace('version 12.', '[^\d]','')"/>
```

Related Topics

match

selected-text

A function that returns the user selected text in the currently active view.

Syntax

```
string = selected-text()
```

Example

```
<alias name="selectedText" value="selected-text()"/>
```

selection

A function that returns the xpath selection of the event object or alias.

Syntax

```
string = selection('[object|alias]')
```

Example

In the example below the alias $currentSelection will contain the selection (if one exists) in the listview1 component located in the view1 UI file.

```
<alias name="currentSelection" value="selection(#view1#listview1)"/>
```

substring

A function that returns a substring as requested by the parameters passed.

Syntax

The function has several parameters. The first being the original string, each following parameter should contain a substring instruction allowing the user to manipulate the original string.

Available instuctions are:

Name	Type	Values	Description
startPos	integer	[0...n]	Defines the staring position of the substring. Used with endPos instruction.
endPos	integer	[0...n]	Defines the end position of the substring. Used with and must the higher than startPos.
startString	string	Any string value	Defines the starting string of the subtring. Used with endString instruction.
endString	string	Any string value	Defines the ending string of the substring. Used with the startString instruction.
startOffset	integer	[0...n]	Defines the offset added to the startPos instruction.
endOffset	integer	[0...n]	Defines the offset added to the endPos instruction.
startType	enumeration	**firstIndex**, lastIndex	Defines if the last occurance of the startString string should be used or the first.
endType	enumeration	**firstIndex**, lastIndex	Defines if the last occurance of the endString string should be used or the first.
substringLength	integer	[1...n]	Defines the length of the substring.

```
string = substring('[string]', '[string]', '[string]?')
```

Example

In the example below we request a substring with the start position of 2 (counting starts with 0) and the end position of 6.

The alias $alphabetExtraction should contain the string CDEFG.

```
<alias name="alphabetExtraction" value="substring('ABCDEFGHIJKL', 'startPos:2', 'endPos:6')"/>
```

trim

A function that returns the passed string but with the leading and the trailing whitespace removed.

Syntax

```
string = trim('[string]')
```

In the example below the alias $newString will contain the string 'hello world'

Example

```
<alias name="newString" value="trim('     hello world      ')"/>
```

trim

A function that returns a string where all occurances of the given substring has been replaced with the second substring. This is identical to the XSLT translate function.

Syntax

```
string = translate('[string]', '[string]', '[string]')
```

In the example below the alias $replacedString will contain the string 'a_b_c'.

Example

```
<alias name="replacedString" value="trim('a b c',' ','_')"/>
```

upcase

A function that returns the passed string in uppercase.

Syntax

```
string = upcase('[string]')
```

Example

In the example below the alias $newString will contain the string 'HELLO WORLD'.

```
<alias name="newSting" value="upcase('hello world')"/>
```

Related Topics

capitalize, downcase

xml

A function that returns a string that contains XML-like contents into XML.

The function will lose its reference to the dataDoc and Xpath selection in the data document passed in as a parameter. This can be put into reference that enclosing an expression with curley brackets "{}" will convert it to a string without any XML.

Syntax

```
xml = xml(string|expression)
```

Example

In the example below we make sure that the component will return an XML node, and not a text string.

```
<alias name="my_value" value="xml(#view1#component1)"/>
```

Related Topics

None available.

Process Operations

This section contains all available operations in XIOS/3.

All operation pages start with a general description of the operation. This is followed by a usage section where we cover how this operation is used and what purpose the attributes and child elements have.

A list of all attributes related to the operation element and then a list of all child elements and their attributes. This is then followed by a section called "Syntax and Examples", were the basic syntax of the operation is followed by examples.

At the bottom of each page is a list of related operations.

action

This operation enables the developer to perform a number of actions on components. All available actions are listed in the "Using Components" section with examples and a list of components that the action can be applied to.

This operation is asynchronous.

Usage

The enable-action enables text areas, input fields and button where as the disable-action disables them. The hide- and show-actions makes components invisible or visible. The select-action emulated the select-event on menus. The release-action emulates the release-event on menus. The unbind-action unbinds any data bound to a component. The clear-action empties fields or text areas from text.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	action	Defines the action operation
value	expression	expression	In what view the components are located

Child Elements

component (Required)

Name	Type	Values	Description
name	string	enumeration	The name of the component on which the action will be performed
action	enumeration	See available actions	The type of action to be performed from the list of

Syntax and Examples

Syntax

```
<operation name="action" value="expression">
  <component name="string" action="enumeration"/>
</operation>
```

Example

```
<operation name="action" value="#view1">
  <component name="abGrid" action="unbind"/>
  <component name="inputName" action="clear"/>
</operation>
```

Example: Process code for the action operation

Related operations

callMethod

bind

This operation enables users to bind data to components thus utilizing the strength with XIOS/3.

Usage

The bind operation is a central operation that is used frequently. The data is either a URL to an XML document or an expression. We recommend that you read the section about Using Expressions to learn more about how to use expressions. The component gains access to that document and will only utilize the bound section of data.

The bound components render the data in its own way. When components are bound to each other they keep track of changes made in either of the components. When the data is changed in one component that same change will happen in the other component.

Binding a component that to another which is not bound to any data will result in that a string of the selected value in the component can be retrieved. This is achieved by evaluating an expression. The reason why we must evaluate the component is because all component are bound to temporary XML data which until we bind data to it.

The model attribute on the component element, an optional attribute, is used to define what data handling to use. Setting it to "none" will concatenate the xpath base path and the select attribute. If it is set to "data" it will instead bind to the root of XML data in the event object, without a base path. A select attribute can be appended to select data relative to the root. "clone" will copy the dataDoc, xpath and selection of the event object (e.g. alias). When it is set to "base" the base path will be copied and the select attribute can be used to select data relative to the base path.

The type attribute, which is also optional, will when set to focus bind the data once and then release the connection. The default value of the attribute allows for the bound data to be changed.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	bind	Defines the bind operation
value	data	expression|URL-to-XML	The XML data to bind

Child Elements

component (Required)

Name	Type	Values	Description
name	string	Any string value	Required. The component to bind the data to
view	string	Any string value	Required. The view where the component is located
select	xpath	xpath	Optional. Relative or absolute path for selection
model	enumeration	default , none, data, clone, base	Optional. Type of data handling to use
transformer	string	URL	Optional. XSL Transformer to change the XML data
type	enumeration	**default**, focus	Optional. Type of data binding to perform
eval	boolean	true, **false**	Optional. The entire XML element is evaluated for expressions if true

paging (child element of component. Optional)

Name	Type	Values	Description
size	integer	1...[n]	Required. Defines the amount of node-sets that will be displayed per page
asc	boolean	**true**, false	Optional. Defines if the results should be bound in ascending order or not
order	string	XPath	Optional. Defines by which element the order should be performed. The value can point to an element or an attribute

Syntax and Examples

Syntax

```
<operation name="bind" value="[expression|xml]">
  <component view="[string]" name="[string]" select="[xpath]"/>
</operation>
```

Example

```
<operation name="bind" value="home://Documents/mydata.xml#/profiles/user[1]">
  <component view="view1" name="component1" select="@name"/>
  <component view="view1" name="component2" select="@city"/>
</operation>
```

Example: Here we bind two different attributes from the first user element contained within mydata.xml

```
<operation name="bind" value="#view1#tree1">
  <component view="view1" name="list2"/>
</operation>
```

Example: Here we bind the data stored inside a component (tree1) to another component (list2)

Related operations

setSelection

call

This operation is used to call other steps.

Usage

Using this operation you can call/evoke a step and also add which operation start the step should begin with. It is a single line element which is controlled using the value attribute which contains id of the step which we intend to call and the step number (separated by a colon).

The operation will call the step which will execute all its operations and then the step that contained the original call operation will resume.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	call	Defines the call operation
value	string	identifier	The step to execute

Child Elements

trigger (optional)

The trigger element allows the calling step to controll the incomming trigger for the step being called. The value attribute defines the sent trigger.

Name	Type	Values	Description
model	string	value or empty	The model attribute can be set to value in order to use the value attribute as plain text to not be parsed as an expression.
value	string	Any string value	Use the value attribute on the element to define the trigger.

yield (optional)

The yield element allows the user interface update before the step is called. The text() node value can be set to either true or false where the default value is true if the element is used.

param (optional)

The param element is used to define the name and value of a local scope alias (variable). The element uses two attributes "name" and "value".

Name	Type	Values	Description
name	string	Any string value	The name of the parameter which can be used in the step called
value	string	Any string value	The value of the parameter

timer (optional)

The timer element is used to define a timer for when the step will be called a single time or over and over again.

Only a single attribute can be used, either **interval** or **wait**

Name	Type	Values	Description
interval	string	1...n, stop	The number of milliseconds between step calls or if the timer should be stopped
wait	integer	1...n	The number of milliseconds until a step should be called

Syntax and Examples

Syntax

```
<operation name="call" value="[string]"/>
```

Example

```
<operation name="call" value="22"/>
```

Example: Process code for the call operation

```
<operation name="call" value="23:3"/>
```

Example: Process code for call step 23 and start executing with the third operation in that step

```
<operation name="call" value="10">
  <yield>true</yield>
</operation>
```

Example: Process code for calling step 10 which allowing the user interface to be updated before the call is performed

```
<operation name="call" value="22">
  <param name="subject" value="Hello"/>
</operation>
```

Example: Process code for calling step 22 and pass a parameter that will be available as an local scope alias

```
<operation name="call" value="22">
  <timer interval="2000"/>
</operation>
```

Example: Process code for calling step 22 after 2000 milliseconds

callMethod

This operation will is used to execute methods on components and even to send parameters to them,

Usage

A list of available methods than can be used can be found in the methods section in "UI Development". Observe that some of the methods listed are not evoked using callMethod but instead appended to an expression.

You might also notice that some methods are executed without any additional parameters while other require that something is placed inside the parameter.

The type attribute on the param element is optional, and the default value for the attribute is string.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	callMethod	Defines the callMethod operation
value	expression	expression	The view or component on which the method should be performed

Optional Attributes

Name	Type	Values	Description
wait	boolean	**false**, true	Wait for asynchronous methods to finish before advancing to the next instruction.
then	string	Any string value	What step to execute once all methods have completed. Good for asynchronous methods without having to stall execution by using wait.

Child Elements

method (Required)

Name	Type	Values	Description
name	string	call	Defines the call operation

param (Child element of method element. Requirement depends on the specific method used.)

Name	Type	Values	Description
type	enumeration	**string**, boolean, integer, float, null, xml, winEvt	Optional. How the contents should be interpreted.

text() (Child node of param element)

The text() node of param contains the information that is to be passed to the component. By default this value is passed on as a white space aware string, but using the type attribute the content can be parsed into something different. The types integer and float are simply parsed as such. The boolean type inteprets the string "true" as true and everything else as false. The xml type will give all its children as a fragment node to the function. The types null and winEvt will ignore its content and in the first case simply send null, and in the second case send the event node from the window, representing the user event that triggered this step to process, if any.

Syntax and Examples

Syntax

```
<operation name="callMethod" value="expression">
  <method name="methodName">
    <param type="[enumeration]">[string|boolean|integer]</param>
  </method>
</operation>
```

Example

```
<operation name="callMethod" value="#view1#component3">
  <method name="setValue">
    <param type="string">hello</param>
  </method>
</operation>
```

Related operations

action

callService

This operations allows XIOS/3 applications to communicate with external web services

Usage

This operation is very complex and does require that you create a channel in order to utilize a web service. The operation does however also support HTTP request using the GET method.

First information must be provided in to the value attribute. It complex URL that needs to be provided contains a way of calling a specific method, a way of uniquely identify the data document using the URL and also define if the web service should be called in a specific interval.

weather://forecast/Linkoping/@60 will contact the channel defined as weather, and execute the "forecast" method. The string "Linkoping" can actually be any string. its primary use is to allow the developer to create unique data documents even though the same exact web service is being used. The last part of our custom URL "@60", instructs the system that this URL (web service call) should be updated every 60 seconds. When loading documents using "/@0", any polling that exists to that specific adress will be stopped.

Observe that if a soapaction is provided as a child element then it "method" set in the value attribute is no longer the method used, instead the value in the soapaction will be called. The document created will still be named after the value in the value attribute of the operation.

The operation will halt your process code by default, however you can change this by adding the **wait** child element as set the text() node to false. This will allow the process code to continue to execute without having to wait for the reponse. It is important to understand that the XML response data is not available until a response is made, so it is generally a good idea to use the default setting which is true.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	callService	Defines the callService operation
value	string	URL	SOAP method, URL identifier and re-connect timer

Child Elements

wait (Optional)

Decides if the process code should continue executing even if no response has been received from the web service. If set to false the process code will continue without having to wait for the process code to continue. This however means that the alias might not contain XML data if no response has been received.

Value can be either true or false. The default value is true.

soapaction (Optional)

If it is present its text() node will override the soapaction in the value attribute of the operation.

< SOAP MESSAGE > (Optional. Only used with soapaction element.)

Actual SOAP message intended for the endpoint of the message

method (Optional. Only used when using HTTP requests)

When using services over http you can specify the method to use; GET

Syntax and Examples

Syntax

```
<operation name="callService" value="[channel://method/id/@numeric]">
  <wait>[boolean]</wait>
  <soapaction>[string]</soapaction>
  <method>[enumeration]</method>
  <SOAP MESSAGE/>
</operation>
```

Example

```

<operation name="callService" value="weather://myDocName/StockholmFC1/@60">
  <soapaction>http://www.webservicex.net/GetWeather</soapaction>
  <GetWeather xmlns="http://www.webservicex.net">
    <CityName>Stockholm</CityName>
    <CountryName>Sweden</CountryName>
  </GetWeather>
</operation>

<!-- debug value of web service document -->
<operation name="debug" value="weather://myDocName/StockholmFC1/@60"/>
```

Example: Process code for the callService operation

```

<operation name="callService" value="http://xcerion.com/example/">
  <wait>true</wait>
  <method>GET</method>
  <param name="firstname">{$name}</param>
  <param name="lastname">{$lastName}</param>
  <param name="email">{$email}</param>
  <param name="from">{#Invite#inputYourName}</param>
  <param name="message">{#Invite#textarea}</param>
</operation>

```

Example: callService using GET method

Related operations

channel, login

change

This operation is used to manipulate XML data and is the key to using the XIOS/3 filesystem

Usage

It offers five different ways of storing data and can also be used to delete data. The five store types are append, replace, replaceText, prepend and insert.

Append will add the node-set supplied in the text() node. Replace will replace the selected node-set with the new node-set supplied in the text() node, while the replaceText will is to only replace text() nodes or attributes. ReplaceText will also create the attribute if did not already exist.

The prepend type will also add the data supplied but unlike the append type it will be placed at the top of the XML document. Insert type is used to insert XML data in to components such as the XIOS/3 filesystem.

Deleting data is done by either deleting the current selection of a component or by supplying an XPath expression in the select attribute.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	change	Defines the change operation
value	expression	expression	The XML data (URL or Component) to which the change should be performed

Child Elements

store (Optional)

Defines that the information should be stored according to the option chosen in the type attribute.

Name	Type	Values	Description
select	xpath	xpath	Optional. Using relative xpath define in which element to store information
transformer	string	URL	Optional. XSL Transformer to change the XML data
type	enumeration	append, insert, move, prepend, replace, replaceText	Required. The type of store to use
value	xpath	xpath	Required for move. The element to move to the selected destination

Type	Description
append	Appends XML data to the end document
insert	Inserts an XML node-set in the document after the current selection
move	Moves an XML node-set from one part of the document to another part
prepend	Appends an XML node-set at the beginning of the document
replace	Replaces the selected XML node-set with the provided node-set
replaceText	Replaces the selected attribute or text()-node with a new value

text() (Child node of store element)

The text() node of store contains XML information or text that is to be appended or that is to replace some other XML or text.

delete (Optional)

Defines that the selected information should be deleted.

Name	Type	Values	Description

select	xpath	xpath	Optional. What type of data is to be sent

Syntax and Examples

Syntax

```
<operation name="change" value="[expression]">
  <store select="[xpath]" type="[enumeration]">[node-set|string|boolean|integer]</store>
</operation>
```

Examples

1. append (Adding XML data)

Will append a node-set to a data document (or atom:feed). The added data will be appended at the bottom of the document. Append can be used to create new files in the XIOS/3 filesystem.

```

<operation name="change" value="home://Documents/">
  <store select="/atom:feed" type="append">
    <atom:entry>
      <atom:title>{#view1#subject}</atom:title>
      <atom:author>{#view1#author}</atom:author>
      <atom:content>{#view1#post}</atom:content>
    </atom:entry>
  </store>
</operation>
```

Example: Process code for the change operation (append)

2. replace (Replace or update XML data)

Will replace an entire node-set

```

<operation name="change" value="#view1#component1">
  <store select="task" type="replace">
    <task>
      <startDate>{#view1#tcStartDate}</startDate>
      <status>{#view1#status}</status>
      <description>{#view1#appRichtext}</description>
    </task>
  </store>
</operation>
```

Example: Process code for the change operation (replace)

3. replaceText (create attributes or replace text() nodes or attributes)

Will replace just the text()-node. This can be used to rename files and folders.

```
<operation name="change" value="#view1#component1">
  <store select="task/@complete" type="replaceText">true</store>
</operation>
```

Example: Process code for the change operation (replaceText)

4. Prepend (Appending XML data)

This type will add the XML data but instead of the bottom of the document it will be added at the top of the document.

```
<operation name="change" value="#view1#component1">
  <store select="atom:feed" type="prepend"><myElement>Will become the first post</myElement></store>
</operation>
```

5. Insert (Inserting XML data in to component)

This type will insert XML data in to a component, such as the RichText.

```
<operation name="change" value="#view1#component1">
  <store type="insert"><P><SPAN STYLE="color:000000;">Hello</SPAN></P></store>
</operation>
```

6. Delete (Removing XML data)

Will delete the selected node-set. If passed a component it will delete the selection, else you need to supply it with a selection.

```

<operation name="change" value="home://Documents/">
  <delete select="/atom:feed/atom:entry[atom:title='{$fileTitle}']"/>
</operation>
```

Example: Process code for the change operation (delete)

7. Move (Move an XML node-set to another part of the document)

Will move letter F (source) and place it before letter C (destination).

```

<alias name="dataDoc" value="#MoveTest"/>

<operation name="create" value="$dataDoc">
  <list>
    <letter>A</letter>
    <letter>B</letter>
    <letter>C</letter>
    <letter>D</letter>
    <letter>E</letter>
    <letter>F</letter>
  </list>
</operation>

<operation name="change" value="$dataDoc">
  <store type="move" select="/list/letter[. = 'C']" value="$dataDoc#/list/letter[. = 'F']" model="data"/>
</operation>
```

Example: Process code for the change operation (move)

Related operations

create

close

This operation is used to close documents from XIOS/3, by clearing them from the cache.

Usage

This operation is used to close applications, XML documents, file listings and even clearing the cache. This is done by supplying the name of the object that we intend to close or the URL of the document or feed that we are using. This is then combined with the mode child element and its text() node to define what we are trying to close.

To close applications the text() node must contain the word "application". For views we use "view" and for process files we use "process". Documents and atom feeds (file listings) we use the "document" option. To clear the xios document cache we simply write "document_cache" and do not supply any value attribute on the operation element.

The "system" option will clear all vital managers (communication and document) and prepare the system for logout. It does however not logout from the system instead it conserves the privacy of the user.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	close	Defines the close operation
value	string	Any string value	Defines what should be closed

Optional Attributes

None Available.

Child Elements

mode (Optional)

This child element is not controlled by attributes but instead with the text() node.

text() (Child node of mode element)

The type of close that should be executed. Available values are "application", "channel" "document", "document_cache", "process", "system" or "view".

Syntax and Examples

Syntax

```
<operation name="close" value="[string]"/>
```

1. Closing view files

```
<operation name="close" value="addUser">
  <mode>view</mode>
</operation>
```

This will close the view called addUser (as defined by the name attribute of the view file).

2. Closing documents or search feeds

```
<operation name="close" value="home://Documents/clients.xml">
  <mode>document</mode>
</operation>
```

3. Clearing the document cache

```
<operation name="close">
  <mode>document_cache</mode>
</operation>
```

Related operations

open

create

This operation is used to create a XML node-set by creating a new object (temporary document).

Usage

We create an object by writing the name of the object in expression form. This requires that a hash sign is placed in front of the name of the object. When calling the object we then call for #objectname.

While we can create empty objects by not adding any XML data to the text() node the most common is to create at least a root element.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	create	Defines the create operation
value	expression	expression	The object that we want to create

Child Elements

text() (Required)

The text() node contains the XML node-set of the newly created object.

Syntax and Examples

Syntax

```

<operation name="create" value="[expression]">
  [XML node-set]
</operation>
```

Example

```

<operation name="create" value="#Clipboard">
  <clipboard/>
</operation>
```

Example: Here we create a temporary #Clipboard that can be used as long as the application is running

Related operations

change

channel

This operation is used to create a communicator that can communicate to a web service.

Usage

When mounting groups we first supply the name of the communicator in the value attribute on the operation element. To mount normal groups the value of this attribute will always be "xios". When mounting subgroups (groups within groups) we use the name of the group communicator instead. In the mount element we define as what the channel will we mounted and then in the text() node the name of the group (or subgroup).

Attributes

Required Attributes

Name	Type	Values	Description
name	string	channel	Defines the channel operation
value	string	string	Communicator to use

Child Elements (for internal channels / groups)

mount (Required. For mounting groups)

Name	Type	Values	Description
name	string	Any string value	Required. Resulting channel name of the group
friendly	string	Any string value	Optional. Resulting channel name of the group

text() (Child node of mount. Required)

The text() node contains the name of the Group that we indent to mount.

Child Elements (for external channels)

name

Name	Type	Values	Description
text() node	string	Any string value	Name identifier of the service

title

Name	Type	Values	Description
text() node	string	Any string value	Welcome message for when service mount has been successfull

friendly

Name	Type	Values	Description

text() node	string	Any string value	Friendly name for when channels has been mounted

protocol

Name	Type	Values	Description
text() node	enumeration	XMLFS, WebServices	Type of protocol to use

soapaction

Name	Type	Values	Description
text() node	string	URL	URL to webservice SOAP action

wsdl

Name	Type	Values	Description
text() node	string	URL	URL to webservice WSDL file

session

Name	Type	Values	Description
text() node	string	?	?

port

Name	Type	Values	Description
text() node	numerical	1-n	Port number of service

interval

Name	Type	Values	Description
text() node	numerical	1-n	Seconds between web service requests

offline

Name	Type	Values	Description
text() node	enumeration	**default**, use-cache	Defines communication behavior when user is offline

primary

Name	Type	Values	Description
text() node	boolean	**false**, true	Defines if the channel is to be identified as the primary channel for the user

language

Name	Type	Values	Description
text() node	string	URL	Path to Protocol XML Schema (XSD file)

namespace

Name	Type	Values	Description
prefix	string	Any string value	Namespace prefix
text() node	string	URI	URI identifier for namespace

Syntax and Examples

Syntax (Internal channel : Group)

```
<operation name="channel" value="[string]">
  <mount name="[string]" friendly="[string]">[string]</mount>
</operation>
```

Syntax (External channel)

```
<operation name="channel" value="string">
  <name>string</name>
  <title>string</title>
  <friendly>string</friendly>
  <protocol>string</protocol>
  <soapaction>string</soapaction>
  <wsdl>string</wsdl>
  <session>enumeration</session>
  <port>integer</port>
  <interval>ineger</interval>
  <offline>enumeration</offline>
  <primary>boolean</primary>
</operation>
```

Example (Internal channel : Group)

```
<operation name="channel" value="xios">
  <mount name="xidezone" friendly="x-zone">XideZone</mount>
</operation>
```

Example: Process code for the channel operation

Example (Internal channel : subGroup)

```
<operation name="channel" value="xidezone">
  <mount name="xidezoneVIP" friendly="xzVIP">XideZoneVIP</mount>
```

```
</operation>
```

Example: Accessing a subgroup of XideZone called XideZoneVIP. Observe that we are using xidezone as the parent communicator

Example (External channel)

```
<operation name="channel">
  <name>https</name>
  <title>HTTPS</title>
  <protocol>HTTPS</protocol>
</operation>
```

Example: Process code for the channel operation

Related operations

login

debug

This operation is used to debug applications in when developing for XIOS/3.

Usage

Several different ways of debugging applications is offered. Two of the debug options utilize the XIOS/3 debug window. When omitting the text() node and when using the "full" option information will be outputted in the debug window which can contain detailed information about components and data. Observe that these are only visible in the debug window and that end-users will not see them. Using the alert or alert:plain options the process code will be halted until you click on the button that appears in the dialog window.

The other two debug options will make debug windows appear. Using "alert" will open a system alert window while the process code will still be executed. Due to this several system dialogs can be displayed at the same time once your processes code is done executing and the last debug window will be on top.

Enabling Debug

To simplify development, XIOS/3 can be turned in to "debug" mode. This is done by activating debug using the Control Panel application, click Development and choose Yes for "Activating debugging traces". You may now toggle the debug window by clicking on the blue icon in the top right corner.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	debug	Defines the debug operation
value	expression|string	expression, Any string value	What we intend to debug

Child Elements

text() (Optional)

While the text() node is optional it is used to control how the debug information is displayed. The allowed values of text() are: "alert", "alert:plain", "full".

Syntax

```
<operation name="debug" value="[expression|string]">[enumeration]</operation>
```

Example

1. debug (in debug window)

```
<operation name="debug" value="Text output in debug window"/>
```

Example: Here the text will be outputted in the debug window

2. debug (system dialog - will halt processcode)

```
<operation name="debug" value="#view#component#/section[1]">alert</operation>
```

Example: XML data output in a XIOS/3 dialog window. The process code will continue to run uninterrupted in the background.

3. debug (JavaScript dialog - will halt processcode)

```
<operation name="debug" value="#view#component#/section[1]">alert:plain</operation>
```

Example: XML data output in a JavaScript-style alert-dialog window. The process code will halt until you click the OK-button.

4. debug (details in debug window)

```
<operation name="debug" value="#view#component">full</operation>
```

Example: Detailed information about the component from the processing language.

decision

This operation defines a switch-statement which is used to test conditions.

Usage

The operation has two elements, "when" and "otherwise". The "when" element is used to test if an XPath condition is true or false. If it is true then the step supplied in the step attribute will be executed. If no step attribute is given, the first statement with a true test will execute the contents of the element.

There is an "exists" attribute that can be used instead of the "test" attribute to test if a specific file in the filesystem exists. This attribute can only be set to "true", i.e. setting it to "false" will not execute the statement if the file does not exists.

Since the decision operation is a switch statement, it will go through all of the "when" tests in order until one of them tests true. If none of them do, the "otherwise" element is unconditionally executed. The placement of the "otherwise" element doesn't matter.

This operation allows nestled logic (operations or aliases). Meaning that you can add operation elements as child elements to the "when" and "otherwise" elements. Once the conditions have been met the operations are executed. The step attribute has higher priority than inline operations meaning that if both an attribute and inline operations exist the attribute is used. It is accetable to not have either an attribute or inline operations which would meet a condition but do nothing.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	decision	Defines the decision operation

Optional Attributes

Name	Type	Values	Description
value	expression|string	expression, Any string value	What we intend to apply tests on

Child Elements

when (Required)

Any operations that should be exeucted can be placed directly within the when element as child elements which are executed once the condition is true.

Name	Type	Values	Description
exists	boolean	true	If a file exists
test	condition	condition	The XPath condition to test
step	string	identifier	The step to execute

otherwise (Optional)

Any operations that should be exeucted can be placed directly within the otherwise element as child elements which are executed once the condition is true.

Name	Type	Values	Description
step	string	identifier	The step to execute

Syntax and Examples

Syntax

```
<operation name="decision" value="[string]">
  <when test="[condition]" step="[string]"/>
  <otherwise step="[string]"/>
</operation>
```

Example

```
<operation name="decision" value="#view1#mails">
  <when test="count(/documents/item) = '0'" step="18"/>
  <when test="count(/documents/item) = '1'" step="19"/>
  <when test="count(/documents/item) = '2'">
    <operation name="debug" value="TWO">alert:plain</operation>
  </when>
  <otherwise>
    <operation name="debug" value="SOMETHING ELSE">alert:plain</operation>
    <operation name="call" value="200"/>
  </otherwise>
</operation>
```

Example: Process code for the decision operation

```
<operation name="decision" value="#view1#comp1">
    <when test="self::*">
      <operation name="debug" value="Element Match!">alert:plain</operation>
    </when>
    <when test="not(self::*)">
      <operation name="debug" value="Attribute Match!">alert:plain</operation>
    </when>
    <when test="*">
      <operation name="debug" value="This element has childnodes!">alert:plain</operation>
    </when>
  </operation>
```

Example: Testing node relationships

filesystem

This operation offers various ways of handling filesystem instructions.

Usage

Each filesystem operation can only have one single child element at any given each. Below is a list of available child elements. Each of these is an instruction that is applied on a file or folder of your choosing. The operation was created to solve much of the issues regarding the filesystem, and to make it easier to manage files.

Automatic renaming is supported by default to make sure that name collisions do not occur.

The **create** instruction is used to create files and has two attributes, "type" and "name". The "type" attribute is an enumeration type attribute and defines to and can be set to "file", more options will follow. The "name" attribute defines the name of the newly created file or folder. You can include several create elements inside the operation.

The **delete** instruction is used to delete files and folders and has two attributes, "permanent" and "silent", both which are boolean type attributes. If permanent is set to true the file will be permanently deleted, and setting it to false will only delete it to the trashcan. The silent attribute controls the end-user interaction, when set to true it will wait until confirmation from the end-user.

The **copy** instruction is used to copy files and folders and has two attributes, "to" and "silent". The "to" attribute is a string type attribute, and is used to define the destination folder. The "silent" attribute is a boolean type attribute and it controls the end-user interaction, when set to true it will wait until confirmation from the end-user. You can include several copy elements inside the operation.

The **move** instruction is used to move files and folders and has two attributes, "to" and "silent". The "to" attribute is a string type attributes, and is used to define the destination folder. The "silent" attribute is a boolean type attribute controls the end-user interaction, when set to true it will wait until confirmation from the end-user. You can only have a single move element inside the operation.

The **rename** instruction is used to rename files and folders. There is only a single attribute "to" which is a string type and should contain the new name of the file or folder.

The **attach** instruction is used to attach files to other files. It has two attributes "to" and "slot". The attribute "to" is a string type attribute, which defines to which file we are trying to attach a document to. The second attribute "slot" is a numeric type attribute and defines the attachment slot to remove. Observe that slot 1 is the actual file, so the first attachment actually has slot number 2. If the slot attribute is omitted the operation will find an unused slot by itself.

The **detach** instruction is used to detach attachments from files. It has only one attribute "slot" which is a numeric type attribute and defines the attachment slot to remove. Observe that slot 1 is the actual file, so the first attachment actually has slot number 2.

The **upload** instruction is used to upload files to a destination folder. It has only one attribute "to" which is a string type attribute and defines the destination folder. Observe that when using the upload instruction the value attribute on the operation element should be omitted.

The **trahscan** instruction is used to deal with the system trashcan. The "value" attribute is used for two different purposes depending on the value in the "action" attribute. If the action is to "empty" the trashcan then "value" attribute should contain the string text "xios". If however the purpose is to "restore" a file then the "value" attribute should contain a string referencing to the xios-url of the file which should be restored. The trashcan element also has a "silent" attribute which allows the developer to define if the end-user should confirm that the trashcan action should be performed.

The **share** instruction is used to share files/documents/photos and folders with friends. Any file or folder which the user has created in his filesystem can currently be selected to be shared. The user may however not share a file which someone has shared with him. The instruction only allows file sharing with people which are found in the user's friendlist (#Friends). This is combined with a access control list which applies to all the users selected. There is also an optional comment which can be passed along the share instruction which can be displayed to the user once he/share takes part of the share information. For the receiving user to files will be visible in his/her filesystem (home://) but the files and folders are still in the control of the user which has shared the files. The files are not copied but infact the user performing the share allows the user receiving the share access to read the files stored on his drive.

The **listen** instruction is used to listen to changes made in a folder or a document. The object to listen to is defined in the "value" attribute of the operation element. A "listen" element is then added as a child element to the operation element. This element has two attributes "event" and "step". The "event" attribute defines the event identifier to fire when the contents of the file or folder have changed. Here you are allowed to define any value. The "step" attribute contains which step to call once the event is fired. Both attributes are required.

The **unlisten** instruction is used to remove a file or folder listener (see above). The object to remove the listener from is defined in the "value" attribute of the operation element. A "unlisten" element is then added as a child element to the operation element. This element has a single attribute "event" which defines which listener to remove specifically. The attribute is required.

The **publish** instruction is used to externally publish files and folders.

The **unpublish** instruction is used to unpublish files or folders that are already published.

The **mount** instruction is used to mount a new filesystem.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	filesystem	Defines the filesystem operation
value	string	Expression	The data to which the operation will be performed

Child Elements

create (Optional)

Name	Type	Values	Description
type	enumeration	file, folder	Required. Defines what is to be created
alias	string	Any string value	Optional. Name of alias to be created containing the new folder as the value
name	string	Any string value	Optional. The name of the file or folder to be created
rename	boolean	**false**, true	Optional. If the file or folder should be autorenamed

author (Child element of create. Optional)

The author element contains information that is placed in the **atom:author** element inside the file feed. The element can have two different elements as child elements, the "name" element which contains the value of "atom:author/atom:name" and the "email" element which contains the value of "atom:author/atom:email".

summary (Child element of create. Optional)

The summary element contains the value that is stored in the **atom:summary** element inside the file feed, and can be used for a short description of the file.

acl (Child element of create. Optional)

The acl (Access Control List) element can contain information about access rights. It can contain **user** or **group** elements which in turn have several attributes.

Name	Type	Values	Description
id	integer	[n]	Required. The id of the user or group

read	boolean	**true**, false	Required. Defines if the user or group has right to read the content of the file or folder
write	boolean	**false**, true	Required. Defines if the user or group has right to write content in the file or folder
delete	boolean	**false**, true	Required. Defines if the user or group has right to delete the file or folder
changeACL	boolean	**false**, true	Required. Defines if the user or group has right to change the rights of the file or folder

index (Child element of create. Optional)

The index element can contain information of the desired meta data that is to be added to the file created. The element has one or more **key** elements which define the meta data element

Note that this only works for files since folders do not support meta data.

key (Child element of index. Optional)

It is important to understand that since file searching (using atom and dc namespaced elements) are done using only the local-name of the elemnent (ie a for the atom:name value "Goran" is done by typing in "name:Goran"). This means that you must NOT name your dc-elements to names already used by the atom specification (eg name, summary, author etc).

Key elements with the index attribute value set to true will be available as "dc:[name]" (eg dc:something) and those with name value false will be available as "ni:[name]" (eg ni:something). The dc-namespaced are **searchable** while the ni-namespaced elements are not.

Name	Type	Values	Description
name	string	Any string value (please read description text above)	Required. The name of the meta data element
index	boolean	Any string value (please read description text above)	Required. Defines if the value should be indexed (searchable) or not

content (Child element of create. Optional)

The content element has different values depending on the value of the type attribute on the create element. If the type value is file then the content folder accepts a type attribute for which file type the file should have, the default value is "text/xml". Inside the content element the entire file content can be added, if the file is an XML file.

If the create element's type attribute value is "folder" then the content attribute contains the potential value of the content attribute on that folder's folder-element in folders.xml.

Name	Type	Values	Description
type	string	File type	Optional. The filetype association of the created file

delete (Optional)

Name	Type	Values	Description
permanent	boolean	true, false	Optional. Defines what is to be created
silent	boolean	true, false	Optional. Defines if the end-user should be asked to confirm

move (Optional)

Name	Type	Values	Description
to	string	URL	Required. Destination folder
silent	boolean	true, false	Optional. Defines if the end-user should be asked to confirm

copy (Optional)

Name	Type	Values	Description
to	string	URL	Required. Destination folder
silent	boolean	true, false	Optional. Defines if the end-user should be asked to confirm

rename (Optional)

The rename instruction is used to rename files or folders. The value attribute of the **operation** element should contain the XIOS URL of the folder or file.

Name	Type	Values	Description
to	string	Any string value	Required. Defines the new name of the file/folder

attach (Optional)

The attach instruction accepts XIOS/3 http URLs.

Name	Type	Values	Description
to	string	URL	Required. Destination file
slot	numerical	2...n	Optional. Attachment slot number. Slot 1 is occupied by actual file

detach (Optional)

Name	Type	Values	Description
slot	numerical	2...n	Required. Attachment slot number. Slot 1 is occupied by actual file

upload (Optional)

When the file is uploaded it will be added to the file feed, and placed in the correct alphabetical order. If there is no sorting order (as might be the case when doing a searchQuery) then the sorting order is defaulted to published.

Name	Type	Values	Description
to	string	URL	Required. Destination folder
to	boolean	**true**, false	Optional. Defines if a flash uploader should be used or not

trashcan (Optional)

Name	Type	Values	Description
action	enumeration	restore, empty	Required. Defines the action performed to the selected file
silent	boolean	**true**, false	Optional. Defines if the user should confirm the action performed

share (Optional)

Required Attributes

Name	Type	Values	Description
useSelection	boolean	**true**, false	Required. Defines if the selected users should be allowed to read the shared file(s)

The share instruction also has child elements, which when used with the value attribute on the filesystem element (the parent of the share element) allows you to share your file with one or more users or groups, to set access control lists and to pass a comment.

The shared files will end up in the home://Friends/ folder under a folder named after the user that shared the files (eg home://Friends/kurtCobain/).

users (Required. Child element of share)

This element should contain friend element nodes in the same format as found in #Friends. Thus the element contains a list of selected friends.

comment (Child element of share. Optional)

The comment element contains a string of maximum 256 characters and should include a message a message which is passed on the the users which the file is shared with.

listen (Optional)

The listen instruction is used to set up a listener for changes in a specific file or folder. Once a change has occured an event is fired and the value of the "step" attribute is executed.

Observe that in order for this to work you need to define the object to listen to in the value attribute of the operation element as a path.

Required Attributes

Name	Type	Values	Description
event	enumeration	New, Change, Append, Prepend, Replace, ReplaceText, RemoveNode, Delete	The file/folder change event to trigger the execution step.
step	string	Any string value	The step to execute once the event has occured

unlisten (Optional)

The unlisten instruction is used to remove listeners that have previously been set up.

Observe that in order for this to work you need to define the object to listen to in the value attribute of the operation element as a path.

Required Attributes

Name	Type	Values	Description
event	string	Any string value	The event identifier to remove

mount (Optional)

The mount instruction is used to mount a new filesystem similarily to how channel sets up a new communication channel. The difference is that a new filesystem is using an already established communication channel.

Required Attributes

Name	Type	Values	Description
name	string	Any string value	The protocol name of the mounted drive.

Required Attributes

Name	Type	Values	Description
friendly	string	Any string value	The display name of the mount.

group (Child element of mount. Optional)

Mount a group.

Required Attributes

Name	Type	Values	Description
name	string	Any string value	The name of the group to mount.

drive (Child element of mount. Optional)

Mount a drive.

Required Attributes

Name	Type	Values	Description
name	string	Any string value	The name of the drive to mount.

Syntax and Examples

Syntax

```
<operation name="filesystem" value="[expression]">
  <instruction/>
</operation>
```

Example: create (File)

```
<operation name="filesystem" value="home://Documents/">
  <create type="file" name="MyNewDocument.xml">
    <author>
      <name></name>
      <email></email>
      <uri></uri>
    </author>
    <summary></summary>
    <acl>
      <user id="101" read="true" write="true" delete="false" changeACL="false"/>
    </acl>
    <index>
      <key name="keywords" index="true"></key> <!-- will be indexed as dc:keywords -->
      <key name="site" index="false"></key>    <!-- will be indexed as ni:site -->
    </index>
    <content type="text/xml">
      <something>My New File...</something>
    </content>
  </create>
</operation>
```

Example: The operation will create a new document called MyNewDocument.xml in the Documents folder.

Examples

Example: create (Folder)

```
<operation name="filesystem" value="home://Documents/">
  <create type="folder" name="Contactbook">
    <content>Contact</content>
  </create>
</operation>
```

Example: The operation will create the folder 'Contactbook' inside the Documents folder. The folder content will be Contact file types.

Example: delete

```
<operation name="filesystem" value="home://Documents/MyDocument.xml">
  <delete permanent="true" silent="false"/>
</operation>
```

Example: The operation will delete the MyNewDocument.xml in the Documents folder, permanently but will ask the end-user to confirm it.

Example: copy

```
<operation name="filesystem" value="home://Documents/MyDocument.xml">
  <copy to="home://Applications/" silent="false"/>
</operation>
```

Example: The operation will copy MyNewDocument.xml in to the Applications folder, end-user will be asked to confirm it.

Example: move

```
<operation name="filesystem" value="home://Documents/MyDocument.xml">
  <move to="home://Applications/" silent="true"/>
</operation>
```

Example: The operation will move the MyNewDocument.xml from the Documents folder to the Applications folder. The end-user will not have to confirm it.

Example: attach

```
<operation name="filesystem" value="http://xios3.com/image.jpg">
  <attach to="home://myFile.xml" slot="2"/>
</operation>
```

Example: The operation will attach an a second attachment (http://xios3.com/image.jpg) to myFile.xml in the Documents folder

Example: detach

```
<operation name="filesystem" value="home://Documents/myFile.xml">
  <detach slot="3"/>
</operation>
```

Example: The operation will detach the third attachment from myFile.xml in the Documents folder

Example: upload

```
<operation name="filesystem">
  <upload to="home://Documents/"/>
</operation>
```

Example: The operation will detach the third attachment from myFile.xml in the Documents folder

Example: trashcan

```
<operation name="filesystem" value="xios">
  <trashcan action="empty" silent="false"/>
</operation>
```

Example: The operation will empty the filesystem trashcan

```
<operation name="filesystem" value="home://Document/myDoc.xml">
  <trashcan action="restore"/>
</operation>
```

Example: The operation will restore the specific document from the trashcan to the folder from which is was deleted

Example: share

```
<operation name="filesystem" value="#view1#gridFilesToShare">
  <share useSelection="true">
    <users>{#view1#gridSelectedFriends}</users>
    <comment>{#view1#commentInputfield}</comment>
  </share>
</operation>
```

Example: The operation will share the selected files with the selected users

Example: listen

```
<operation name="filesystem" value="home://Documents/">
  <listen event="FileAdded" step="550"/>
</operation>
```

Example: The operation will listen to changes in the home://Documents/ folder and execute step 550 do perform some logic

Example: unlisten

```
<operation name="filesystem" value="home://Documents/">
  <unlisten event="FileAdded"/>
</operation>
```

Example: The operation will remove the listener we created in the example above

foreach

This operation allows you to iterate an XML node set and execute operations while iterating.

Usage

To use the foreach operation you first need to provide an XML document, which could be either a URL, a plain document reference (ie #document) followed by an XPath pointing to node-set which should be iterated or it could be an Expression which would cause the operation to iterate over the selections. The complete expression string should be provided in the value attribute.

Inside the operation element you can add the operations that should be iterated in the for each process. The iterated node-set is available in the trigger data (ie the exclamation mark - !). From that you can create new expressions to obtain the data from the node-set.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	foreach	Defines the foreach operation
value	string	expression | parameters	Defines the data document and which node to iterate or iteration parameters

Optional Attributes

None Available.

Child Elements

operation (Required)

Any process operation.

Syntax and Examples

Syntax

For iteration of XML nodes, such as for each atom:entry element in an atom:feed.

```
<operation name="foreach" value="[Xpath]">
  <!-- process XML code that should be executed for each item-->
</operation>
```

Example: foreach process operation

For iteration by incrementation, such as for 1 to 10 with an incrementation of 2.

```
<operation name="foreach" value="[int..int], int">
  <!-- process XML code that should be executed for each iteration-->
</operation>
```

Example: foreach process operation

The first two integers define the start and endindex. The third integer is optional and defines the incrementation and has the default value of 1.

Example

```
<step id="1" name="Iteration step">
  <!-- iterate over file listing -->
  <operation name="foreach" value="home://Documents/#atom:feed/atom:entry">
    <operation name="debug" value="Document {!#atom:title}">alert:plain</operation>
  </operation>

  <!-- iterate over selections -->
  <operation name="foreach" value="#view1#listview1">
    <operation name="debug" value="Document {!#atom:title}">alert:plain</operation>
  </operation>
</step>
```

Example: The operation will iterate over a file listing and over set of selections

The trigger (!) contains the iterated node-set, which in this case is an atom:entry node set.

Example 2 - Interval Iteration

```
<operation name="foreach" value="12..25,3">
  <operation name="debug" value="!">alert:plain</operation>
</operation>
```

Example: Iteration for a start index to an end index with intervals

The code above will start iterate from 12 to 25 with the interval 3. The value of the ! (trigger) will contain the current index value ie: 12, 15, 18, 21, 24.

Related operations

None available.

friends

This operation allows you to perform actions on friends or friend groups in the #Friends document (the user's list of friends).

Usage

The operation works by adding a instruction inside the operation element that defines which action should be performed and to which user/group it should be performed.

The **create** instruction allows for creation of friend groups (roles).

The **remove** instruction allows for removal of friends and/or friend groups.

Attributes

Required Attributes

None available

Optional Attributes

None available

Child Elements (instructions)

create

Instruction for creating friend groups.

Attributes

Name	Type	Values	Description
name	string	Any string value	Name of friend group to create

remove

Instruction for removing friends and friend groups.

Attributes

Name	Type	Values	Description
group	string	Any string value	Name of friend group to remove
user	integer	userId	Removes a friend with a specific userId

groups

This operation allows you to perform action and receive information from groups.

Usage

By using child elements as instructions you will be able to perform any supported command. Each command is given as an instruction, of which there can be more than one.

The **accept** instruction will allow a users to accept a group invitations, which in turn is done from inside the group by a member or an administrator.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	groups	Defines the login operation
value	string	*	Context attribute, value depends on each instruction used. See separate instructions below for details

Optional Attributes

None available

Child Elements (instructions)

accept

Attributes

Name	Type	Values	Description
to	string	group path	Absolute path to group from which the user is accepting the group invitation

addAvatar

The **addAvatar** instruction is used to save a group specific avatar for the user.

This instruction also uses the **value** attribute of the operation element to define the URL to the image, which to use.

Attributes

Name	Type	Values	Description
height	numerical	0...[n]	Defines the height of the selected image
left	numerical	0...[n]	Defines the left edge of the image to start cropping from
to	string	group path	Absolute path to group from which teh user is accepting the group invitation
top	numerical	0...[n]	Defines the top edge of the image to start cropping from
width	numerical	0...[n]	Defines the width of the selected image

deleteAvatar

This instruction is used to delete the current avatar for a specific group. It does not utilize the **value** attribute. It will delete the ni:avatar element from the user's atom:entry data (user's group profile data).

The group path is for first level groups just the group name. For second level groups (groups beloning to groups) must provide the full path (ie 'firstGroupName/subgroupName').

Attributes

Name	Type	Values	Description
group	string	groupPath	The path of the group for which the avatar should be deleted

isPublic

This instruction also uses the **value** attribute is used to define which group to check.

Attributes

Name	Type	Values	Description
alias	string	Any string value	The name of the alias that will contain the returned value (true or false)

Syntax

```
<operation name="groups">
  <instruction to="[string]"/>
</operation>
```

Example (accept)

```
<operation name="groups">
  <accept to="home://Groups/{$group}"/>
</operation>
```

Example (addAvatar)

```
<operation name="groups" value="{$dropSrc}">
  <addAvatar to="xios" left="0" top="0" width="128" height="128"/>
</operation>
```

Example (isPublic)

```
<operation name="groups" value="home://Groups/XideZone">
  <isPublic alias="publicGroup"/>
</operation>
```

login

This operation allows you to connect to password protected web services

Usage

Login requires the inbound data to be equal to the data used that is used when a communicator is created. We listen to the "Login" event from the new communicator. The trigger (!) available when the event is caught will be equal to the channel operation and it is this information that the login operation uses to access the communicator.

An example of how the trigger-element used to catch the event might look like can be found at the bottom of this page.

We then proceed to add a username and password in the text() nodes of the respective child elements.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	login	Defines the login operation

value	string	Any string value	The communicator to use

Child Elements

username (Required)

The username is supplied in the text() node. No attributes are available.

password (Required)

The password is supplied in the text() node. No attributes are available.

reset (Optional)

If the current connection should be reset. This is an optional element. No attributes are available.

Syntax and Examples

Syntax

```
<operation name="login" value="[node-set]">
  <username>[string]</username>
  <password>[string]</password>
  <reset>[boolean]</reset>
</operation>
```

Example

```
<operation name="login" value="!">
  <username>{#Login#Username}</username>
  <password>{#Login#Password}</password>
</operation>
```

Example: The operation will login to the channel alias

Trigger element used to capture the Login event from the newly created xios communicator, which will make the information in the channel operation available as (!):

```
<trigger communicator="xios" event="Login" step="20"
```

Related operations

channel, callService

navigate

This operation is used to nagivate in XML documents

Usage

Navigate uses the expression supplied in the value attribute. Looking at the event model we expect the expression to contain an URL to the XML document, an xpath and a selection. More about expressions can be found in the " Using Expressions" section.

It is by using navigate that we can change the selection while the data making the new information available in the trigger (!). Observe that the selection will not have been made on the component which the data came from. Using this we can keep track of which file is "next" (following) and which is "previous" (preceding) relative to the current file.

Note: In the case of multiple selections the axis will be applied to all of the selections.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	navigate	Defines the navigate operation
value	expression	expression	XML node-set to navigate

Child Elements

call (Required)

Name	Type	Values	Description
axis	enumeration	parent, preceding, following, first, last	which axis to nagivate to
element	string	Any string value	If the navigation should be limited to an element (without namespace)
step	string	identifier	The step to execute

Syntax and Examples

Syntax

```
<operation name="navigate" value="[expression]">
  <call axis="[xpath-axis]" step="[string]"/>
</operation>
```

Example

```
<step id="15" name="Goto Previous Operation">
  <operation name="navigate" value="!">
    <call axis="following" step="10"/>
  </operation>
```

```
</step>
```

Example: When this step is called it will jump to the parent of the selected node in the tree-component and then execute step 10.

open

This operation is used to open process, view and application files.

Usage

A URL to the file that we are opening must be supplied in the value attribute. Then there are a number of optional child elements that can be used when opening new files, some are specific to process files and some to view files. Observe that application files count as both so process and view (if such files exist in the application package).

When dealing with process files, we can pass it information. The trigger element contains the trigger that we wish to send to the new process files. The step element contains a reference to the starting step. If no value is supplied here it will try to open step 1. The context element we can be set to true which will allow the new process file to share aliases. Context also allows you to send parameters to the process files that is being called by adding param elements, which then if context is set to true makes aliases from calling process inherited instead of shared. Each param element has a name attribute which is mandatory since parameters are retrieved in the new process file using this name as $paramname. Set context to false if you do not want the default context sharing within an application package and its processes.

If you need to open a process, have aliases shared across them, and allow the new process to execute and have access to the running view with the same expressions to access UI components as in the original context, you need to set the context element to true and provide the trigger element with a value attribute set to "!", in addition to the step element with the id attribute of step to call in the new process. This is very useful when modularizing code for different sections of the application into multiple process files.

When dealing with view files can can decide what window mode should be used by using the window element. While the window mode is the default it can also be set to open in fullscreen mode using the fullscreen value.

NOTE: In order to open a view or application correctly it needs to be opened after bootup logic has been applied such as cathing the trigger data (!) and saving it as an alias.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	open	Defines the open operation
value	string	URL	The file/website/application we intend to open

Optional Attributes

Name	Type	Values	Description
secure	boolean	**false**, true	If the https protocol should be used

Child Elements

param (Optional)

Several parameters can be added and are available as aliases in the process file opened. Parameters can be sent to Process XML files and Application Files.

Name	Type	Values	Description
name	string	Any string value	The name of the parameter
value	string	Any string value	Value of the parameter

name (Optional)

No attributes available only text() node is used.

position (Optional)

Places the opened view at the given position, unless there is a saved window state for the user.

Name	Type	Values	Description
x	integer	Positive or negative integer	Set the horizontal position of the view
y	integer	Positive or negative integer	Set the vertical position of the view
rel	expression	expression	View or component to be relative against. Relative against upper left corner. If used x and y will be added to the relative position.

trigger (Optional)

Name	Type	Values	Description
model	enumeration	**data**, value	Defines the incomming trigger for the step to be executed, use "!" to allow the opened process step to continue and execute on the same context as the calling step. This will allow expressions to be evaluated against the calling UI view or channel.
value	string	URL	Trigger data to use

step (Optional)

Name	Type	Values	Description
id	string	Any string value	Defines starting step

browser (Optional)

Name	Type	Values	Description
name	string	Any string value	Component identifier
secure	boolean	true, false	if secure http should be used
features	string	Any string value	Properties separated with a specific syntax

Note: The features attribute has the following syntax: "property=value, property=value"

The following properties are available: height (int), width (int), status (boolean), toolbar (boolean), location (boolean)

window (Optional)

No attributes available only text() node is used. Available values: "window" (default), "fullscreen", "main".

action (Optional)

No attributes available only text() node is used. Available values: "open" (default), "preview", "edit".

context (Optional)

No attributes available only text() node is used. Available values: true, false, clone. If element and no values are supplied, default will be true if process is opened within an application package, otherwise false. If set to clone a copy of the context will be used.

focus (Optional)

This element regulates if the window should be focused when opened. If set to false the window will be opened as minimized.

No attributes available only text() node is used. Available values: true (default), false.

handleClose (Optional)

This element will allow the application the fire the "Closing" event when trying to close a window allowing the developer to decide what will happen when the "X" is clicked. If true is set the application will not close when the "X" is clicked.

No attributes available only the **text()** node is used. Available values: false (default), true.

Syntax and Examples

Syntax

```
<operation name="open" value="[string]">
  <name>[string]</name>
  <trigger value="[URL]"/>
  <step id="[string]"/>
  <param name="[string]" value="[string]"/>
  <window>[enumeration]</window>
  <action>[enumeration]</action>
  <context>[boolean]</context>
  <focus>[boolean]</focus>
  <handleClose>[boolean]</handleClose>
</operation>
```

Example 1

```
<operation name="open" value="dashboard/processes/cli.xml">
  <context>true</context>
  <trigger value="!"/> <!-- This transfers the current context into the called step -->
  <step id="executeCommand"/>
</operation>
```

Example: Modularize process code into several process files, have the new process execute within the same context as the calling process.

Example 2

```
<operation name="open" value="xios/apps/shell/shell.xml">
  <step id="1"/>
  <param name="subject" value="Hello"/>
  <param name="message" value="This is a longer parameter that is send using operation open"/>
  <context>true</context>
  <focus>true</focus>
  <handleClose>true</handleClose>
</operation>
```

Example: Open shell application and start with first step

Related topics

close

print

This operation is used to print documents.

Usage

The default configuration of the operation will print using a default XSL that will transform the XML data to HTML. For this no **transform** element needs to be defined just the **value** attribute which contains a reference to the data document to print.

If another xsl transformer should be used then it can be defined using the transform element, by setting the **type** attribute to "xsl" and the **value** attribute to the URL pointing to the xsl transformer.

If there is a need to print plain XML then the type attribute of the transform element to "xml".

Attributes

Required Attributes

Name	Type	Values	Description
name	string	print	Defines the print operation

value	string	Expression	Defines which data/document to print

Optional Attributes

Name	Type	Values	Description
debug	boolean	true, **false**	If true the output will open in a new tab instead of being printed.

Child Elements

transform (Optional)

This element is used to configure how the data should be printed.

Attributes

Required Attributes

Name	Type	Values	Description
type	enumeration	**html**, xml, xsl	Defines to what the data should be transformed

Optional Attributes

Name	Type	Values	Description
value	string	URL	Defines which XSL transformer to used when type is xsl

Syntax and Examples

Syntax

```
<operation name="print" value="[Expression]">
  <transform type="[enumeration]" value="[string]"/>
</operation>
```

Example

```
<operation name="print" value="{#view#richtext.getValue()}"/>
```

Example: This will perform a print on the contents of a richtext component

```
<operation name="print" value="#Help#content">
  <transform type="xsl" value="apps/myapp/transformers/specialprinter.xsl">
    <style href="xios/apps/help/layout/help.css"/>
  </transform>
</operation>
```

Example: This will perform a print on the contents of thge trigger using a custom transformer

rollBack

This operation is used to rollback transactions.

Usage

The Transactionmanager (#TransactionManager) keeps track of all transactions made in the session and by using the rollBack operation we can revert the document to a previous state. This is possible since all transactions are stored locally during the session.

The rollBack operation allows you to rollback the changes made to a certain document by providing the URL to that specific document in the value attribute. Doing this will trigger a rollback only for that document and will not affect any other transactions every if they occurred after the last transaction in the specific document.

If we make a reference to the transactionManager (#TransactionManager) in the value attribute and it has a selection, it will roll-back to the selected transaction. To only limit the rollBack to a certain transaction of a certain document you need to provide the documents child element with the text() value "current". This will inform the system that the rollBack should only be performed on a single document, leaving all other transactions intact.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	rollBack	Defines the rollBack operation
value	expression	expression	Defines the XML that contains all the changes made

Child Elements

model (Optional)

This element does does not have any attributes and only uses the text() node. Available values are: (default), data. Where "data" removes the selection and sets the xpath to the root.

documents (Optional)

This element does not have any attributes and only uses the text() node. Available values are "current". Where "current" informs the system that the rollBack should only effect the file currently selected in the #TransactionManager.

Syntax and Examples

Syntax

```
<operation name="rollBack" value="[URL|Expression]"/>
```

Example

```
<operation name="rollBack" value="#TransactionManager">
  <documents>current</documents>
</operation>
```

Example: This will perform a rollBack of the originator file of the selected transaction in the TransactionManager, to the current selected transaction

setSelection

This operation is used to set selections to components that are bound to data.

Usage

Using this operation one can predefine a selection thus emulating the selection of an end-user. Because of this the setSelection might trigger the Select event.

The operation works with the following components:

Calendar, Editor, Grid, ListView, Media, Tree

Attributes

Required Attributes

Name	Type	Values	Description
name	string	setSelection	Defines the setSelection operation

Child Elements

component (Required)

Name	Type	Values	Description
view	string	Any string value	Name of view where the component is located
name	string	Any string value	Name of the component which the operation is to be performed on

item (Child element of component. Required)

Name	Type	Values	Description
select	xpath	xpath	Relative or absolute xpath selection

Syntax and Examples

Syntax

```
<operation name="setSelection">
    <component view="[string]" name="[string]">
      <item select="[xpath]"/>
    </component>
</operation>
```

Example

```
<operation name="setSelection">
    <component view="Explorer" name="folderSelector">
      <item select="/fs:folder/fs:folder[1]"/>
    </component>
</operation>
```

Example: Process code for the decision operation

Related operations

bind

save

This operation is used to save XML documents.

Usage

Saving documents on the XIOS/3 filesystem is done using this operation. To learn how to create files see "Managing Files" section. A reference to the file saved in placed in the value attribute. To make a "save as" dialogue appear an child element named "saveas" must be added.

The saveas functionality is useful when creating new XML node sets using create or alias model new.

When a document is saved, the meta data of the document will be updated.

Note: When working with with a file the document must be saved. Even though a document might appear to to saved without having to save it, that is just because it has been transaction handled however it has not been saved by the server. Next time a new copy of the document is loaded it will load the last saved document.

When working with meta data and file listings you do not need to save the changes as the framework will treat this data specially and save it automatically.

Attributes

Required Attributes

Name	Type	Values	Description
name	string	save	Defines the save operation
value	string, expression	string, expression	The file to save

Child Elements

saveas (Optional)

No attributes available. The saveas element will trigger a "Save As" dialog.

Syntax and Examples

Syntax

```
<operation name="save" value="[string|expression]"/>
```

Example

```
<operation name="save" value="home://Documents/myfile.xml"/>
```

Example: Process code for the open operation

```
<operation name="save" value="$dataDoc">
  <saveas/>
</operation>
```

Example: Saving a file with a save dialog

```
<operation name="save" value="$dataDoc">
  <saveas/>
  <folder>home://Documents</folder>
  <title>Save Presentation</title>
</operation>
```

Example: Saving a file with a save dialog with pre determined folder and dialog title

Related operations

close

share

This operation is used for actions related to Window Sharing and Gesture Based Collaboration.

Usage

view

The **value** attribute on the operation element is used to define which view to share. If the attribute is not provided then the selected view will be automatically chosen.

The **view** element has three attributes which define how the window sharing will be initiated. The **action** attribute defines if the session is to be initiated (singleton or clone) or if it should be ended (end). When initiated the option "singleton" means that both users share a single window which can be dragged across the two users screens, allowing only a single user to work at the document. When "clone" mode is used then both users will see the window simultaneously and both can work at the document at the same time.

When initiating a share window session then the attibutes **user** and **direction** are used. The user attribute defined to whom the window should be shared, while the direction attribute defines in which direction the window should be shared (this is only relavant when the share window mode is 'singleton' since only a single window is shared across both screens).

zone

Currently the operation is used to define a user's friends to a specific sharebar side.

Attributes

Required Attributes

None Available

Optional Attributes

Name	Type	Values	Description
value	string	view:instance	Defines which and instance view to share (Only when using the view element)

Child Elements

view

This element is used when sharing application windows between other users.

Required Attributes

Name	Type	Values	Description

action	enumeration	singleton, clone, end	Defines which action to perform. end will terminate the session while singleton and clone will initiate it

Optional Attributes

Name	Type	Values	Description
direction	enumeration	left, right	In which direction to window should be shared (Only for singleton mode)
user	string	Any string value	The user to which the window should be shared

zone

This element is used when defining a share zone, that is to define that a friend should be positioned at a specific sharebar.

Note: This is required in order for Window Sharing to work.

Required Attributes

Name	Type	Values	Description
direction	enumeration	right, left	Defines in which sharebar the friend should be positioned
user	string	Any string value	A user from the #Friends document to positioned in the sharebar

Optional Attributes

None Available

Syntax and Examples

Syntax

```
<operation name="share">
  <zone direction="[enumeration]" user="[string]"/>
</operation>
```

Example

```
<operation name="share">
  <zone direction="right" user="goran"/>
</operation>
```

split

This operation is used to split a string in to an XML node-set

Usage

By supplying the operation a string in the value attribute of the operation element we can split the string by almost any token. Observe that while expressions are allowed in the value attribute that they must be evaluated to strings so that they become strings. This is done writing the term "use" in the text() node of the value child elements. By setting this to use expression in the value attribute of the operation is interpreted as text and not an expression.

The document element is the name object created and then used when referencing to the new xml node-set. The token element contains the token that will be used to split up the string. Observe that all of these do not have any attributes; instead their values are placed inside the text() node.

The node-set is available as the value provided in the document node and is an XML document.

Tip: If you are trying to use space as the split token then you must place the space inside a CDATA section like such:

```
<token><![CDATA[ ]]></token>
```

Attributes

Required Attributes

Name	Type	Values	Description
name	string	split	Defines the split operation
value	string	string or evaluated expression	String to split

Child Elements

value (Required)

No attributes. Value supplied in text() node. Available values: use.

document (Required)

No attributes. Value supplied in text() node.

token (Required)

No attributes. Value supplied in text() node.

Syntax and Examples

Syntax

```
<operation name="split" value="[string]">
  <value>[enumeration]</value>
  <document>[string]</document>
  <token>[string]</token>
</operation>
```

Syntax of XML node-set returned

```
<split>
  <item>value</item>
  <item>value</item>
  <item>value</item>
</split>
```

Example

```
<operation name="split" value="{$time}">
  <value>use</value>
  <document>#arrTime</document>
  <token>:</token>
</operation>
<alias name="Hours" value="{#arrTime#/split/item[1]}"/>
<alias name="Minutes" value="{#arrTime#/split/item[2]}"/>
```

Example: Time alias is split up using the colon as token. We then create two separate aliases (Hours and Minutes) for the splitted time string. The first item in split will contain the time and the second will contain minutes.

Dictionary

A dictionary of terms used in this documentation. Some terms are specific and thier explanation applies when developing in XIOS/3.

atom namespace

The atom namespace is a W3C standard.

It is used frequently when working with the XIOS/3 filesystem, due to the reason that all XIOS/3 file listings are returned as atom feeds.

Namespace declaration

xmlns:atom="http://www.w3.org/2005/Atom"

Reserved atom elements (as defined by the Atom specification)

atom:author

atom:category

atom:content

atom:contributor

atom:id

atom:link

atom:published

atom:rights

atom:source

atom:summary

atom:title

atom:updated

atom:email

atom:entry

atom:feed

atom:generator

atom:icon

atom:logo

atom:name

atom:subtitle

atom:uri

CSS Syntax

CSS is used to override the components default style by using the notation key:value; repeatedly

Content-Type

MIME value describing the content of the file.

Content-Type	Description
text/html	HTML
text/plain	Plain Text
image/gif	GIF Image
image/jpeg	JPEG Image
image/png	PNG Image
video/mpeg	MPEG Video
audio/wav	WAV Audio
audio/mpeg	MP3 Audio
video/mov	Quicktime Video
video/quicktime	Quicktime Video

video/x-ms-wmv	Windows Media Video
audio/x-ms-wma	Windows Media Audio
audio/x-realaudio	RealPlayer Audio/Video (.rm)
audio/x-pn-realaudio	RealPlayer Audio/Video (.ram)
video/x-msvideo	AVI Video
video/avi	AVI Video
application/pdf	PDF Document
application/msword	MS Word .doc file
application/zip	ZIP File

data transaction

A change in XML data that is bound to a component.

dc namespace

The dc namespace (Directory) is a namespaced used by XIOS/3 to markup which meta data should be indexed and searchable.

Namespace declaration

```
xmlns:dc="http://xcerion.com/directory.xsd"
```

fs namespace

The fs namespace is used when handling folders in the XIOS/3 filesystem

Namespace declaration

```
xmlns:fs="http://xcerion.com/folders.xsd"
```

l namespace

The l namespace is used when handling language localizations of applications.

Namespace declaration

```
xml:l="http://xios3.com/lang"
```

local-name

Local-name refers to the element when the namespace is omitted.

Element without namespace (local-name)

```
<myElment/>
```

Element with dc namespace

```
<dc:myElment/>
```

mutex

Mutual Exclusion. A state when only one component (e.g. panel) can be visible at any given time.

MIME type

See Content-Type for more information.

ni namespace

The ni namespace (No Index) is a namespace used by XIOS/3 to markup which meta data should NOT be indexed and searchable.

Namespace declaration

```
xmlns:ni="http://xcerion.com/noindex.xsd"
```

text()

The text() is a XML node that is equal to the text between the starting and closing XML element.

Example

```
<myElement attr="val">Welcome to the Jungle</myElement>
```

In the above example the following is considered to be the text() node = "Welcome to the Jungle".

tool-tip

Tool-tip is a text that appears when hovering over a component.

xlink namespace

The atom namespace is a W3C standard.

It is used when working with xlink (xlink:actuate, xlink:href and xlink:type).

The namespace declaration should be present on all files in all application packages and any file that utilizes xlink functionality.

Namespace URI

http://www.w3.org/1999/xlink

Namespace declaration

xmlns:xlink="http://www.w3.org/1999/xlink"