# *XUL*

The future of
user-interfaces
on the web

# About me

- Stefan Neufeind
- From Neuss (near Düsseldorf, Germany)
- Working for SpeedPartner GmbH
  in consulting, development and administration

# Agenda

- Background / history
- Basics
- Simple widgets / elements
- Scripting
- Trees, RDF, templates
- Events
- XUL with PHP
- Resources

# Skills needed

Basic knowledge about:

- HTML / XML
- CSS
- JavaScript
- DOM

Easy to start with, but still quite much to learn

# Tools needed

- Mozilla-based browser as runtime
- Some texteditor :-)

for more than easy examples:

- Javascript-console
- Javascript-debugger (“Venkman”)
- Component-viewer
- DOM-inspector
- Docs and references

# Mozilla's way

- 1998: Netscape opened source-code to public
- Code hard to maintain
- Multi-platform-basis incomplete for license reasons
- Mozilla-project reinvented almost complete backend
  - Layout-engine Gecko
  - XPFE (cross-platform frontend)
    - Targeted at applications, not websites
    - Based on XUL (XML user-interface language)

# Mozilla, XUL and ... Ghostbusters?

- XUL: reference to the film Ghostbusters

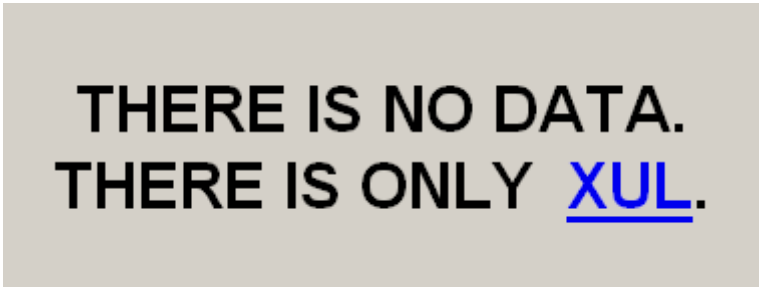
  ghost of an ancient Sumerian deity called Zuul
  (pronounce to rhyme with “cool”)

- possesses the character Dana Barrett and declares
  "There is no Dana, only Zuul"

  XUL usually to define interface, not a document,
  so developers adopted slogan "There is no data, only XUL"

# Mozilla, XUL and ... Ghostbusters?

- XML-namespace-URI:
  http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul
  View with XUL-capable application:




- "Keymaster" and "Gatekeeper" also references to same plotline
- More references to Ghostbusters within Mozilla:
  e.g. JavaScript debugger “Venkman” is one of main characters

[thanks to Wikipedia]

# Big picture

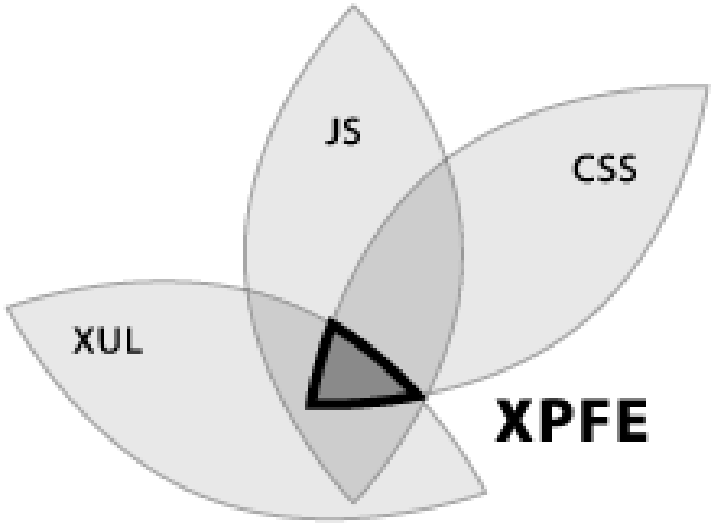
- XPFE: cross-platform frontend
- eXtensible Binding Language (XBL)
  to create reusable widgets with XUL/JavaScript.
- Chrome
  provides user interface parts of application
- XPCOM/XPConnect
  allow JavaScript (and other scripting languages) to access/utilize C and C++ libraries
- Cross-Platform Install (XPInstall)
  to package applications for installing on any platform

# Big picture

- XML-based User-interface Language (XUL)
  to create the structure and content of application
- XUL Templates
  to create framework for importing data with RDF/XUL
- Resource Description Framework (RDF)
  to store data and transmit information
- Document Type Definition (DTD)
  used for localization and internationalization

# What is XUL?

- Markup-language
- Objects as XML-structures
- Elements in a DOM-tree
- Modification of element-attributes possible at runtime
- Styling using CSS




- C++-APIs accessible using XPCOM

# XML basics

- Start with XML-header
- Case-sensitive
  (window != Window)
- Close all tags!
- “Empty” tags allowed in some cases; closed immediately
  <label value=”Hello” />
- Well-formed, unknown tags/attributes ignored
  Allows usage of additional features
- Include namespaces!
  Multiple namespaces supported in one document

# XUL file structure

- Well structured XUL-file contains:
  - XML-version
  - Reference to style-sheets
  - DOCTYPE (omitted in some cases)
  - Namespace-declarations in root-tag (HTML-namespace optional)

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns=
 "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"/>
```

# Layout basics

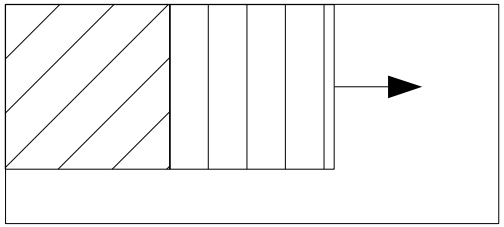
- For XPFE application root of document: <window>
- <dialog> and <page> also roots (recent XUL specs)

“Boxed layout”:

- Basic element is a “box”, contains nested boxes
- Size of “box” dependent on size of parent element
- Layout dependent on axis (“orientation”) and direction (“dir”)

# Layout basics

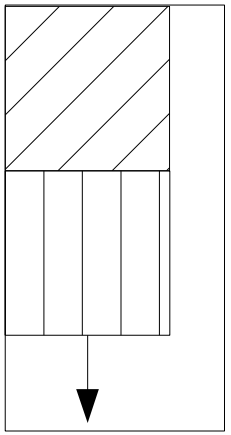
- Horizontal orientation
  - Standard
  - Sub-elements aligned horizontally after each other
  - Complete width: Sum of widths of sub-elements
  - Height: Maximum height of sub-elements



```
<box orientation="horizontal">

                        or in short:

<hbox>
```

# Layout basics

- Vertical orientation
  - Sub-elements aligned vertically after each other (“stacked”)
  - Width: Maximum width of sub-elements
  - Complete height: Sum of heights of sub-elements



```
<box orientation="vertical">

                        or in short:

<vbox>
```

# Layout basics

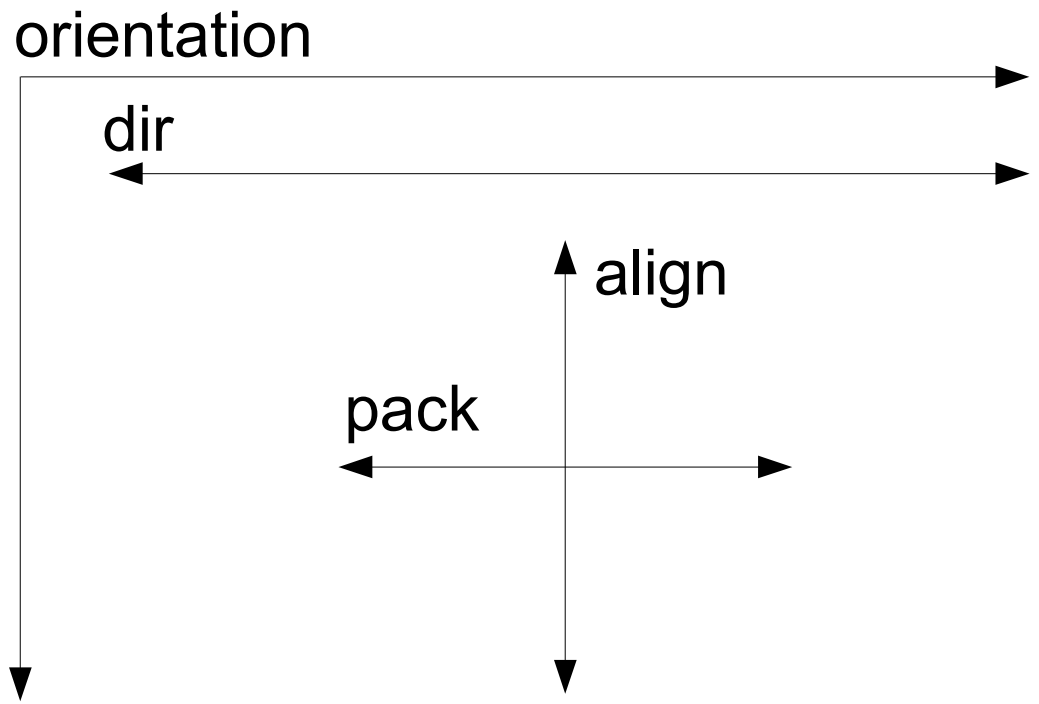
- Direction
  - dir=”ltr” (standard)
    - For hbox: left-to-right
    - For vbox: top-to-bottom
  - dir=”rtl”
    - For hbox: right-to-left
    - For vbox: bottom-to-top

# Layout basics

- Position of child-elements (along primary axis)
  - pack=”start” (position at beginning of parent)
  - pack=”center” (position in middle of parent)
  - pack=”end” (position at end of parent)
- Alignment (along transverse axis; cross-ways to pack)
  - align=”stretch” (Sub-elements stretched parent width/height)
  - align=”start”
  - align=”center”
  - align=”end”:

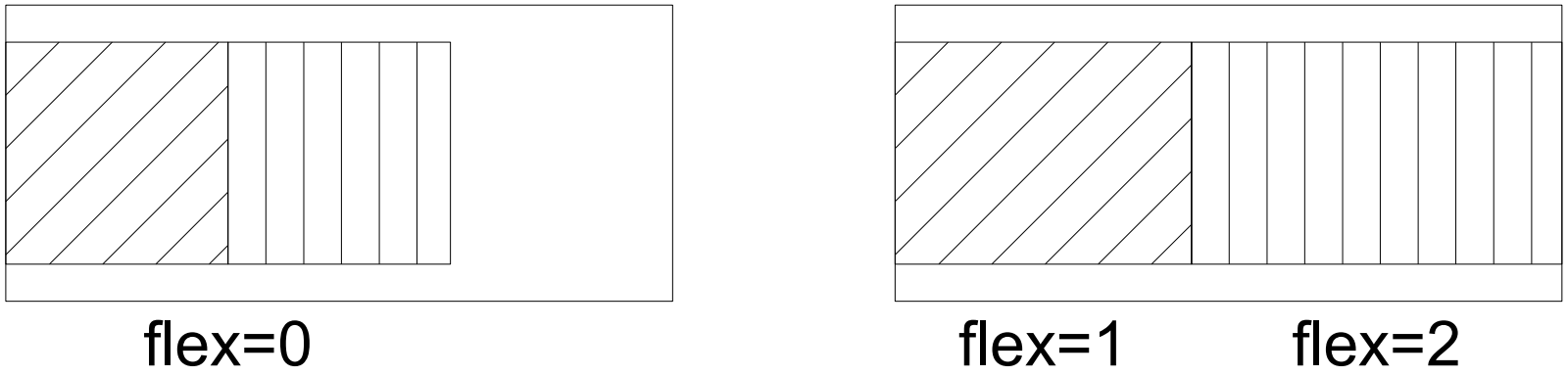
# Layout basics

Standard-attributes of a XUL-layout (“boxed layout”)



# Layout basics

- Attribute “flex”:
  Distribution of additional size for sub-elements
  - Sub-elements with flex>0 are enlarged to fit parent
  - If multiple sub-elements flex>0 exist,
    additional size distributed relative to flex-value
  - Works where align=”stretch” or no align set



# Layout basics

- Size of sub-elements can be set equal
  - equalsize=”always”
  - equalsize=”never” (Standard)
- Only affects sub-elements with flex>0

# Layout basics

- Attributes “height” and “width”
  - Allow giving absolute height/width
  - Might be overridden in case “flex” is used
  - Restrictions possible, that even “flex” obeys
    - minwidth, maxwidth
    - minheight, maxheight

# Layout basics

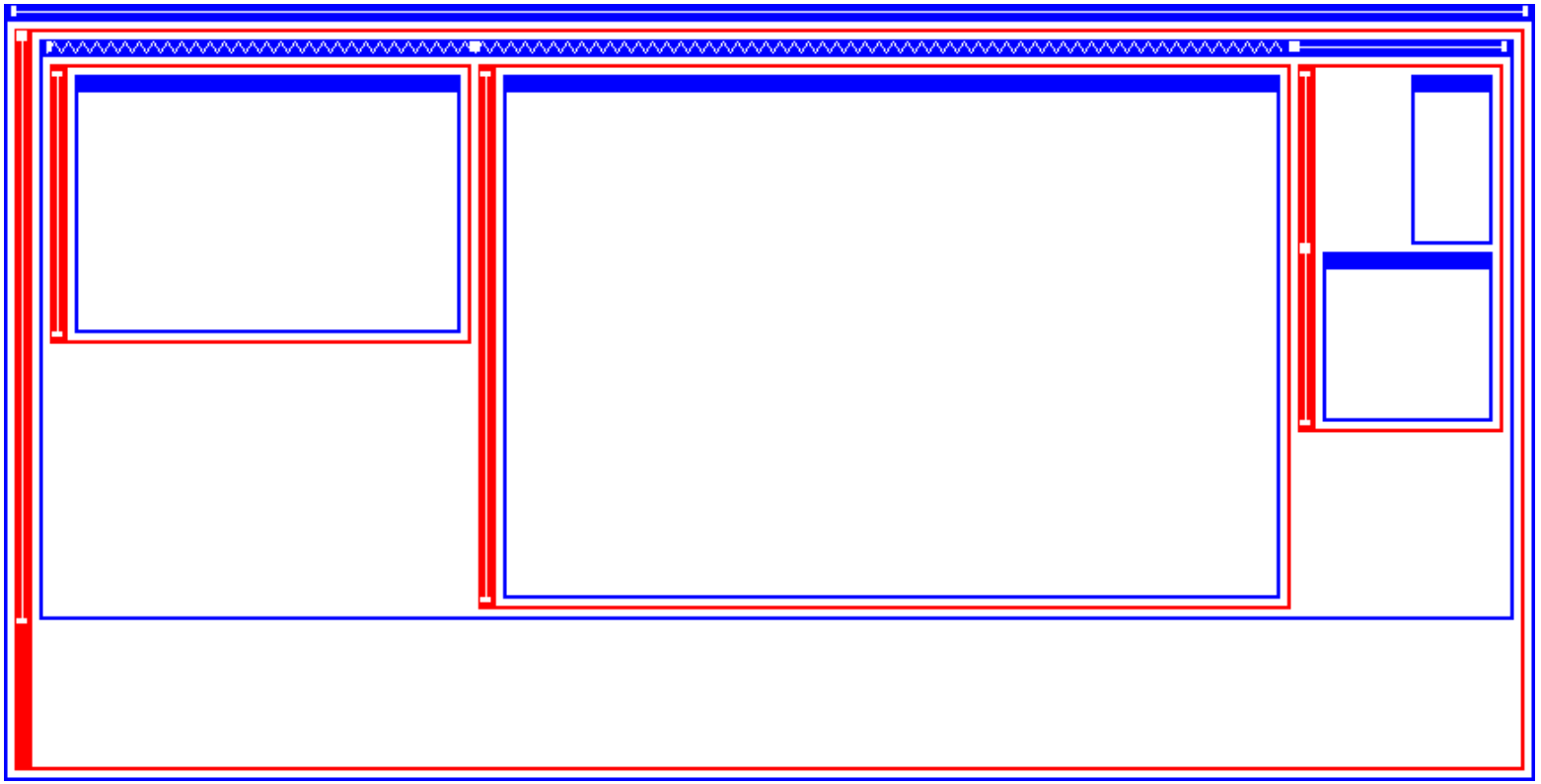
- Root-element for XUL-document needed
- Simplest root-element is an application-window
- Allowed attributes:
  - title (window-title)
  - Only for top-windows (not in browser):
    - width
    - height
    - screenx, screeny  (position on screen)
    - sizemode  (can be “normal”, “minimized” or “maximized”)

# Layout basics

```
<?xml version="1.0"?>
<window xmlns=
 "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
 debug="true">
  <hbox align="start">
    <vbox flex="1">
      <box width="150px" height="150px"/>
    </vbox>
    <vbox flex="2">
      <box width="300px" height="300px"/>
    </vbox>
    <vbox align="end">
      <box width="50px" height="100px"/>
      <box width="100px" height="100px"/>
    </vbox>
  </hbox>
</window>
```

# Layout basics

- Boxes not visible by default
- Using visual debugging: debug=”true” in <window>



# Opening XUL-window

- Open via URL in your browser
  - Displays in tab like webpage
  - URL-inputfield shown, Mozilla-menu available

  - or -

- Open as top window when starting up

```
./mozilla -chrome http://www.example.com/myfirst.xul
```

  - Your XUL-window as top-window
  - Your menu as top-menu

# Styling with CSS

- CSS-usage similar like in HTML
- Using external CSS-files (strongly recommended)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="test2.css"?>
<window xmlns=
 "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"/>
```

- Embedded using data-URIs

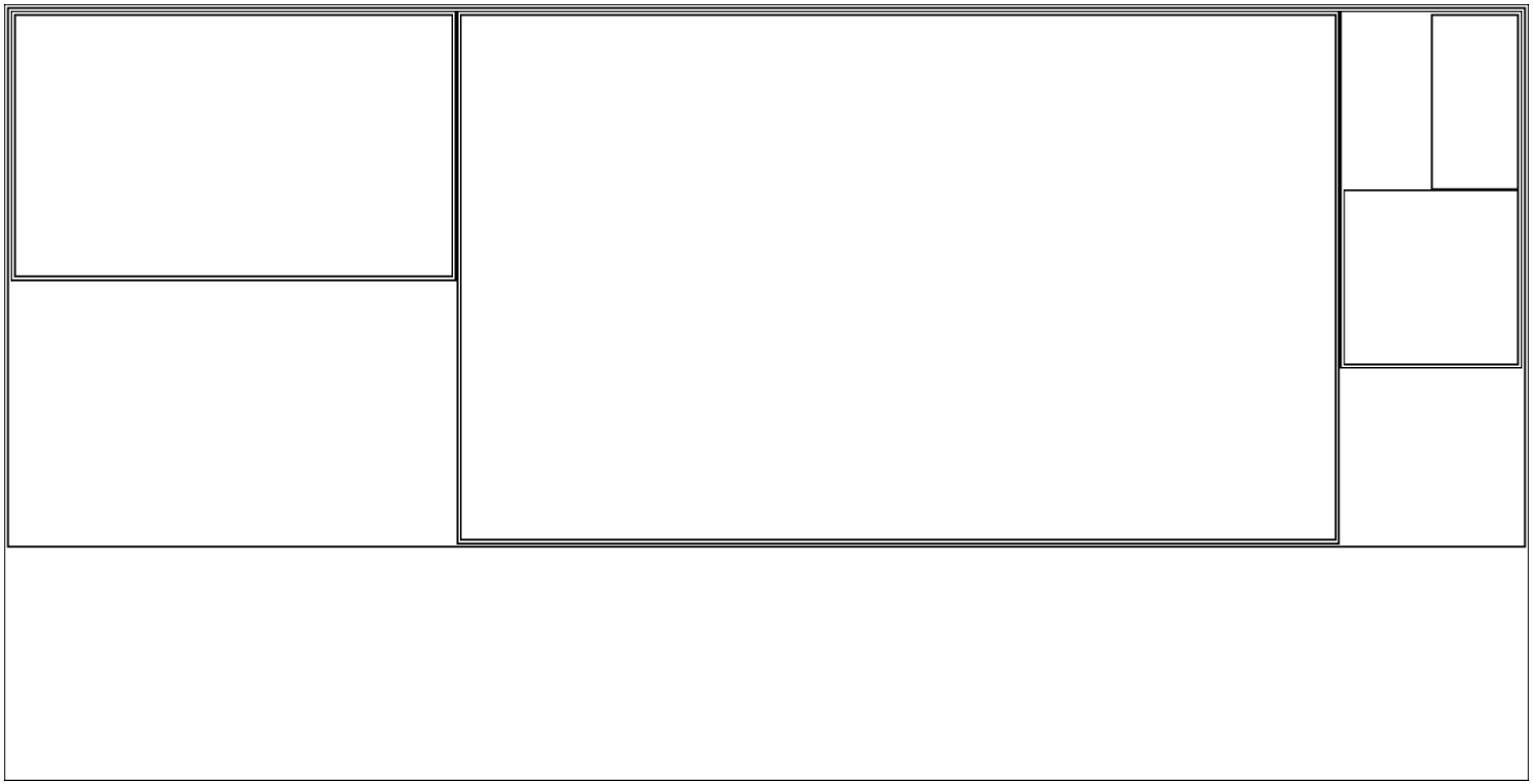
```
<?xml-stylesheet type="text/css" href="data:text/css,
  * {border:1px solid black; padding: 1px;}"?>
```

- Inside element-tags


```
<button label="Button1" style="color:red;"/>
```

# Styling with CSS

- Previous example with CSS-borders



# data-URIs

- Normally URIs point to a resource/file
- RFC2397 defines scheme “data:”

data:[<mediatype>][;base64],<data>

- Mozilla supports data-URIs, e.g. in href=”...”
- Try in browser's location-bar:

```
data:application/vnd.mozilla.xul+xml,<button xmlns=
"http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
label="Click me"/>
```

# HTML inside XUL?

- Not recommended ... but possible :-)
- Using XHTML-namespace explicitly
- Use well-formed XHTML
  (lowercase tags, remember to close tags, ...)


```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"
  type="text/css"?>
<!DOCTYPE window>
<window xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns=
  "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<button label="XUL-button"/>
<html:input type="button" value="HTML-button"/>
</window>
```

# Output of text

- Multiline-text:

```
<description>some ... text</description>
```

- Singleline-text:

```
<description value="some ... text" />
```

# Output of text

- Used as “name-tag” for other elements (buttons, input, ...)

```
<label>some ... text</label>
```

- Not enough space for text?
  Attribute “crop” allows using a crop-”ellipse”
  - “none” reserves needed width for label; does not crop
  - “start” crops at start of label (e.g. “...zilla”)
  - “center” crops label in the middle (e.g. “Mo...la”)
  - “end” crops at end of label (e.g. “Mozil...”)

# Images

- XUL supports web-formats JPEG, GIF, PNG
- Syntax similar to HTML <img> element:

```
<image src="foo.png" />
```

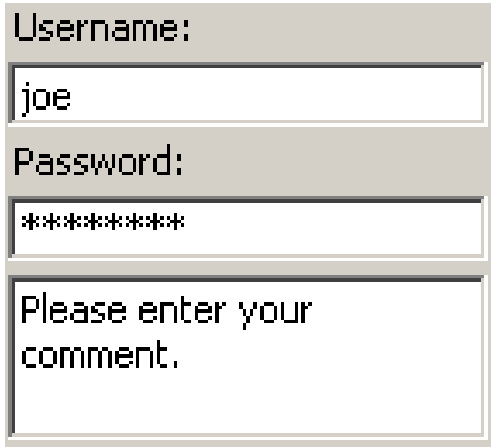
- Or use CSS, property pointing to a resource-URL (e.g. chrome)

```
<image id="foo" />



#foo  {
    list-style-image: url("myImage.png");
}
```

# Text-inputs

- XUL: <textbox> similar to
  HTML: <input type=”text” ...> or <textarea>

```
<label control="username" value="Username:"/>
<textbox id="username"/>
<label control="password" value="Password:"/>
<textbox id="password" type="password" maxlength="20"/>

<textbox value="Please enter your comment." multiline="true"/>
```



# Text-inputs

- Attributes:
  - cols, rows, maxlength (chars allowed to enter), size (chars that can be entered)
  - multiline, wrap (can be “off”, usually on), readonly, disabled
  - type: autocomplete, password, timed (typing-timeout-event)
  - timeout (timeout for type “timed”)
  - onchange (when changing focus away), oninput (immediately during char-input)
- type=”timed”
  - Command event fired after char-input and timeout expired

# Focusing / Shortcuts

- Usually: Using TAB-key moves focus to next displayed element
- Focus moves from last to first element: called “focus ring”

- Possible to define different navigation order:

```
<button label="Button1" tabindex="4"/>
<button label="Button2" tabindex="2"/>
<button label="Button3" tabindex="1"/>
<button label="Button4" tabindex="3"/>
```

Normal order: 1 2 3 4


With tabindex: 4 2 1 3

# Focusing / Shortcuts

- “accesskeys” for selecting a label
  - Usually either underlined by browser or printed after label
  - Allows selection of label by ALT+<accesskey> (Win + Linux)
- “control” gives focus from label to another element

```
<label accesskey="A" control=”otherElement”>
  Accesskey-test
</label>
```

# Scripting

- Mozilla-products use extended version of JavaScript
- Experimental binding for languages like Python exist but not of “real importance” yet
- XUL-documents reflected using a DOM-tree (Document Object Model)
- Additionally internal interfaces available, depending on rights granted to a script
- Scripts possible using event-tags of a XUL-element or “script”-elements

# Scripting

- Using external scripts

```
<script src="toolbox.js" type="application/x-javascript"/>
<script src="chrome://examples/toolbox.js"/>
<script src="http://www.example.com/toolbox.js"/>
```

- Using “embedded” scripts (enclosed in CDATA-block)

```
<script type="application/x-javascript">
<![CDATA[
  function ShowAlert(text)
  {
    window.alert(text);
  }
]]>
</script>
```

- “type”-attribute optional, but highly recommended

# Scripting

- Use id="..." to identify elements in DOM (for getElementById() - explained later)
- Events similar to HTML
  - General: onclick, onmouseover, onkeypress
  - For "window": onload, onunload, onclose
  - For "button": oncommand

```
<?xml version="1.0"?>
<window xmlns=
  "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  title="event basics" id="mainwindow"
  onload="alert('Loaded')"
  onunload="alert('Unloaded')" />
```

# Buttons

- Clickable XUL-element using “button”
  - Element and all sub-elements are one clickable button
  - Allows flexible definition of button-content
  - All known attributes like “orientation” or “dir” supported
  - Usually “label” used to define text on a button
  - Possible to use tag “image” for graphics
  - Attributes “label” and “image” to element “button” automatically create sub-elements inside the button

# Buttons

```
<?xml version="1.0"?>
<window xmlns=
  "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <script type="application/x-javascript">
  <![CDATA[
    function ShowAlert(text)
    {
      window.alert(text);
    }
  ]]>
  </script>
  <hbox>
    <label value="Label1" accesskey="L" control="myButton" />
    <button id="myButton" label="Button1" accesskey="B"
    oncommand="ShowAlert(this.nodeName)" />
  </hbox>
</window>
```

# Buttons

- Sub-elements inside a “button”-element

```
<button id="myButton" accesskey="B"
 oncommand="oncommand(this.nodeName)">
  <label value="LabelB1" />
  <label value="LabelB2" />
</button>
```

- In HTML “onclick” or “onkeypress” would be used
- For button “oncommand” available as well and will also be triggered using accesskey of label or button

# Buttons

- Special button: image with label
- image above, below, left or right of label
  - dir=”normal”  Image is left or above label
  - dir=”reverse”  Image is right or below label
  - orient=”horizontal”  or  orient=”vertical”


```
<button label="Bookmarks" image="http://www.example.com/foo.png"
  dir="reverse" orient="horizontal"/>
```

# Additional element-attributes

- checked=”true”
  Inverts initial state; element is selected, checked, active
- disabled=”true”
  Element is not available for user-interaction (grayed out)
- hidden=”true”
  Element not displayed / included in layout
  Equivalent to “display: none” in CSS
- collapsed=”true”
  Element not displayed, but included in layout
  Equivalent to “visibility: collapse” in CSS

# Checkboxes

- Similar to checkboxes in HTML
- Use XUL-widgets instead of HTML where possible


```
<checkbox id="closeWindow"
  label="Close this window when download is complete"
  checked="true" />
```

# Radiogroups

- One item selectable at a time
- Here used with a small script


```
<script>
function updateState() {
  var name = document.getElementById("name");
  var sindex = document.getElementById("group").selectedIndex;
  if (sindex == 0) name.disabled = true;
  else name.disabled = false; }
</script>
<radiogroup id="group" onselect="updateState();">
  <radio label="Random name" selected="true"/>
  <hbox>
    <radio label="Specify a name:"/>
    <textbox id="name" value="Jim" disabled="true"/>
  </hbox>
</radiogroup>
```

# Groupboxes

- Group elements

```
<groupbox>
  <caption label="The Foo Bar Brothers"/>
  <description value="Hi, this is Foo"/>
  <description value="Hi, this is Bar"/>
</groupbox>
```


- Also “complex” captions allowed

```
<groupbox>
  <caption>
    <checkbox label="Subscribe me"/>
  </caption>
  <hbox>
    <label control="email" value="Email:"/>
    <textbox id="email"/>
  </hbox>
</groupbox>
```

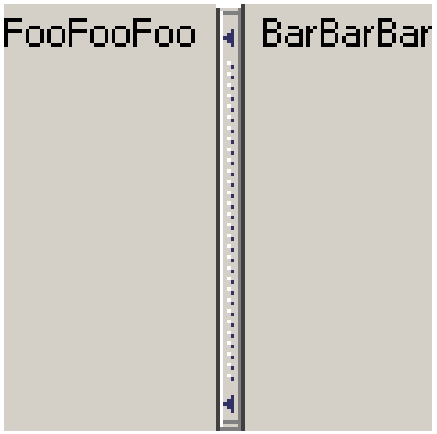
# Grids

- Similar to HTML-table; but no output – just tabular layout
- Rows or columns can be defined


```
<grid flex="1">
  <columns>
    <column flex="3"/>
    <column flex="1"/>
  </columns>
  <rows>
    <row>
      <button label="OSCON"/>
      <button label="Tutorial"/>
    </row>
    <row>
      <button label="Mozilla"/>
      <button label="XUL"/>
    </row>
  </rows>
</grid>
```

# Splitters

- Splits a window/box
- Allows resizing of sections
- Horizontal or vertical, depending on parent box
- Can be collapsed using “grippy” (optional)




```
<description value="FooFooFoo" crop="end"/>
<splitter id="splittername" state="open" collapse="before"
 resizebefore="closest" resizeafter="closest">
  <grippy/>
</splitter>
<description value="BarBarBar" crop="end"/>
```

# Splitters

- Attributes:
  - state “open” (default) or “collapsed”
    Initial state
  - collapse “before”, “after”, “none”
    Side of panel that should collapse when grippy is clicked
  - resizebefore “closest” (default), “farthest”
    Element to resize before splitter
    (farthest = element farthest away from splitter)
  - resizeafter “closest” (default), “farthest”
    Element to resize after splitter

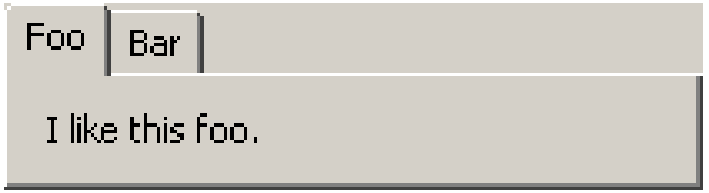
# Tabs

- Displaying set of “subpages” in one window
- Surrounded by a “tabbox”
- “tab”: refers to clickable name-tag of a “subpage”
- “tabpanel”: holds the “subpage”-content

```
<tabbox>
    <tabs>
      [... tab-elements go here ...]
    </tabs>
    <tabpanels>
      [... tabpanel-elements go here ...]
    </tabpanels>
</tabbox>
```

# Tabs

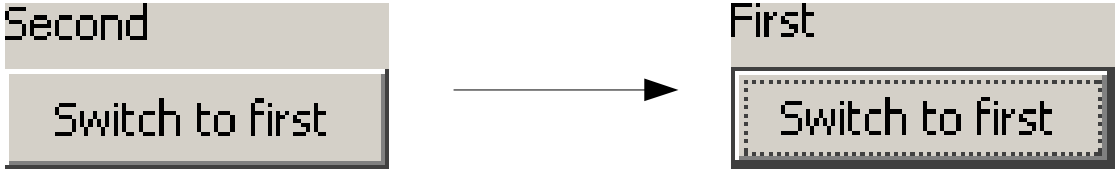
- Easy example:




```
<tabbox flex="1">
    <tabs>
        <tab id="tab1" label="Foo" selected="true"/>
        <tab id="tab2" label="Bar"/>
    </tabs>
    <tabpanels>
        <tabpanel id="tab1Panel" orient="vertical" >
            <description value="I like this foo."/>
        </tabpanel>
        <tabpanel id="tab2Panel" orient="vertical">
             <description value="Let's meet at the bar!"/>
        </tabpanel>
    </tabpanels>
</tabbox>
```

# Decks

- All children displayed at same position
- Only one child displayed at a time
- Displayed deck selectable via DOM (selectedIndex)




```
<deck id="myDeck" selectedIndex="1">
  <description value="First"/>
  <description value="Second"/>
</deck>

<button label="Switch to first"
oncommand="document.getElementById('myDeck').selectedIndex = 0;"/>
```

# Stacks

- Places child-elements on top of each other, back to front
- Useful when selectively showing elements or stacking elements for layout reasons

```
<stack>
    <description value="striked"/>
    <description value="-------"
     style="color:red; font-weight:bold"/>
</stack>
```

striked

```
<stack>
  <description value="shadow"
   style="padding-left:3px; padding-top:3px;"/>
  <description value="shadow"
   style="color:red;"/>
</stack>
```

shadow

# Tooltips / Popups

- tooltiptext="..."
  Text to display when hovering an element
- tooltip="..."
  Takes id of popup to display as tooltip
- Popups allow more complex information than text-tooltips

```
<popup id="helpPopup">
  <description value="some text"/>
</popup>

[...]

<button id="myButton" label="myButton" tooltip="helpPopup"/>
```

# Tooltips / Popups

- Popups usually collected in a “popupset”
- Can be used to offer context-menues
- Position of popups can be given with “position”-tag
  - position=“before_start”  display before element, start-aligned
  - Positions available:
    at_pointer, after_pointer, after_start, after_end,
    before_start, before_end, end_after, end_before,
    start_after, start_before, overlap (would be start_start)

# Tooltips / Popups

```
<popupset>
  <popup id="contextPopup" position="before_start">
    <description value="some text"/>
  </popup>
</popupset>
[...]
<button id="myButton" label="myButton" context="contextPopup"/>
```

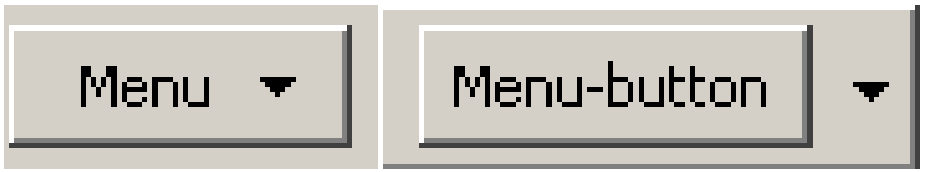
# Advanced buttons

- Special buttons selectable with attribute “type”
  - type=”checkbox”
    button pushed down on first click, released on second
  - type=”radio”
    from group of buttons only one button active at a time
    (attribute “group” identifies group of buttons)

```
<hbox>
  <button type="radio" label="me"  group="whom" />
  <button type="radio" label="you" group="whom" />
  <button type="radio" label="everybody" group="whom" />
</hbox>
```

# Advanced buttons

- Special buttons selectable with attribute “type”
  - type=”menu” or type=”menu-button”
    special marker indicates button will open a menu
    - “menu”: Menu opened upon button activation
    - “menu-button”: User must click marker besides button




```
<hbox>
  <button type="menu" label="menu"/>
  <button type="menu-button" label="menubutton"/>
</hbox>
```

# Toolbars / menus

- Toolbars / menus:
  - Simplified: boxes containing buttons
  - May have “popups” as submenues

- Element “menu” (button, opens a menupopup), contains:
  - “menupopup”
    Container for menucontents
  - “menuitem”
    similar to a “button”
  - “menuseparator”

# Toolbars / menus

- Attributes allowed in <menu ...> and <menuitem ...>:
  - accesskey="..."
  - oncommand="..."
  - disabled="..."
  - type="..." ("radio" or "checkbox")
  - checked="..." (for items of type "radio" or "checkbox")
  - class="..." (anonymous CSS-classes)

  (like with buttons - as you might have already guessed)

# Toolbars / menus

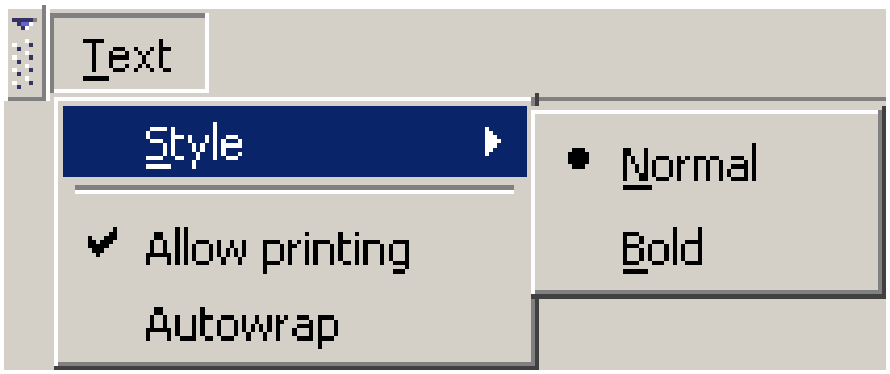
“menupopup”-element:

- Attribute “popupanchor”
  Anchor of parent-“menu”-element to position relative to
  - Values: bottomleft (standard), bottomright, topright, and topleft
- Attribute “popupalign”
  Position of menu relative to anchor
  - Values: bottomleft, bottomright, topright, and topleft

- Good choice for a bottom taskbar might be:
  popupanchor=”topleft” popupalign=”bottomleft”

# Toolbars / menus

- Special menu: “menubar”
  - Only use one “menubar” per window!
    (Mac displays menubar in a special way)
  - For menus in menubar, popupanchor-values except “bottomleft” ignored because of typical space constraints

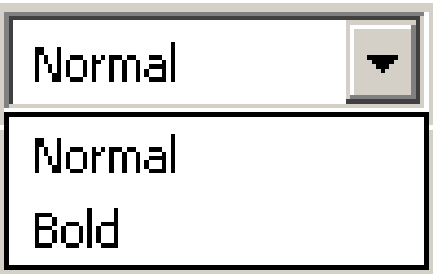


# Toolbars / menus

```
<menubar>
  <menu label="Text" accesskey="T">
    <menupopup>
      <menu label="Style" accesskey="S">
        <menupopup>
          <menuitem label="Normal" accesskey="N" type="radio"
                    name="styleRadio" checked="true"/>
          <menuitem label="Bold" accesskey="B" type="radio"
                    name="styleRadio"/>
        </menupopup>
      </menu>
      <menuseparator/>
      <menuitem label="Allow printing" type="checkbox"
        checked="true"/>
      <menuitem label="Autowrap"/>
    </menupopup>
  </menu>
</menubar>
```

# Toolbars / menus

- Special menu: “menulist”
  Drop-down list




```
<menulist>
  <menupopup>
    <menuitem label="Normal"/>
    <menuitem label="Bold"/>
  </menupopup>
</menulist>
```

# Toolbars / menus

- “toolbar” is a special type of box
- Toolbars usually inside a “toolbox”-container
- Toolbars have “toolbargrippy” besides
  - Allows to collapse them (give toolbars id=”...” for this to work)
  - Automatically created
  - Not supported in Firefox - use Mozilla



2 toolbars collapsed

# Toolbars / menus

```
<toolbox>
  <toolbar id="firstToolbar">
    <button label="Button One"/>
    <button label="Button Two"/>
  </toolbar>
  <toolbar id="secondToolbar">
    <button label="Button Three"/>
    <button label="Button Four"/>
  </toolbar>
  <toolbar id="thirdToolbar">
    <button label="Button Five"/>
    <button label="Button Six"/>
  </toolbar>
</toolbox>
```

# Scrolling

- Special container “arrowscrollbox”
  - Allows adding sub-elements than can be displayed
  - Shows scrollbuttons and performs scrolling


```
<arrowscrollbox orient="vertical" flex="1">
  <button label="Red"/>
  <button label="Blue"/>
  <button label="Green"/>
  <button label="Yellow"/>
  <button label="Orange"/>
  <button label="Silver"/>
  <button label="Gold"/>
  <button label="Black"/>
</arrowscrollbox>
```

# Lists

- List of items
- One or more items selectable
- Number of items adjustable
- Value for each item possible



```
<listbox>
  <listitem label="Red"/>
  <listitem label="Green"/>
  <listitem label="Blue"/>
  <listitem label="White"/>
</listbox>
```

```
<listbox rows="2" seltype="multiple">
  <listitem label="Red" value="r"/>
  <listitem label="Green" value="g"/>
  <listitem label="Blue" value="b"/>
  <listitem label="White" value="w"/>
</listbox>
```

# Lists

- Lists with multiple columns
- Header-row possible
- Sort-indicator possible (does not sort – needs to be done with scripting)

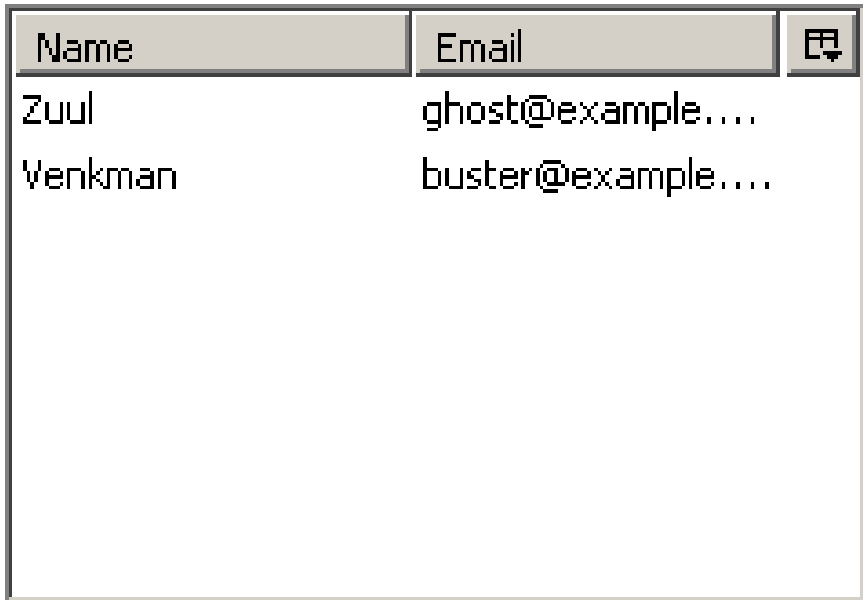

```
<listbox rows="3">
  <listhead>
    <listheader label="Color"
     sortDirection="ascending"/>
    <listheader label="Vehicle"/>
  </listhead>
  <listitem>
    <listcell label="Green"/>
    <listcell label="Boat"/>
  </listitem>
  <listitem>
    <listcell label="Red"/>
    <listcell label="Car"/>
  </listitem>
</listbox>
```

# Trees

- Similar to “listbox” (in a way)
  - Multiple columns and rows
  - Column headers
- Listbox: any type of content / Tree: only text and images
- But offers more advanced features
  - Nested rows
  - Expanding / collapsing of rows
  - ...

# Trees

- Simple tree:
  - Set of columns (treecol)
  - Content (treechildren)
    - Container for rows (treeitem)
      - Rows / Cells




- “Column-picker” to select cols to show

# Trees

- Attributes for a tree:
  - “Column-picker” hideable

```
<tree hidecolumnpicker="true" />
```

  - Usually multiple rows selectable, but single-select possible

```
<tree seltype="single" />
```

- Give id=”...” to every column (!!!)
  - Otherwise showing/hiding columns, selections etc. might now always work

# Trees

```
<tree flex="1">
  <treecols>
    <treecol id="Name" label="Name" flex="1"/>
    <treecol id="Email" label="Email" flex="1"/>
  </treecols>
  <treechildren>
    <treeitem>
      <treerow>
        <treecell label="Zuul"/>
        <treecell label="ghost@example.com"/>
      </treerow>
    </treeitem>
    <treeitem>  <treerow>
        <treecell label="Venkman"/>
        <treecell label="buster@example.com"/>
    </treerow>  </treeitem>
  </treechildren>
</tree>
```

# Trees

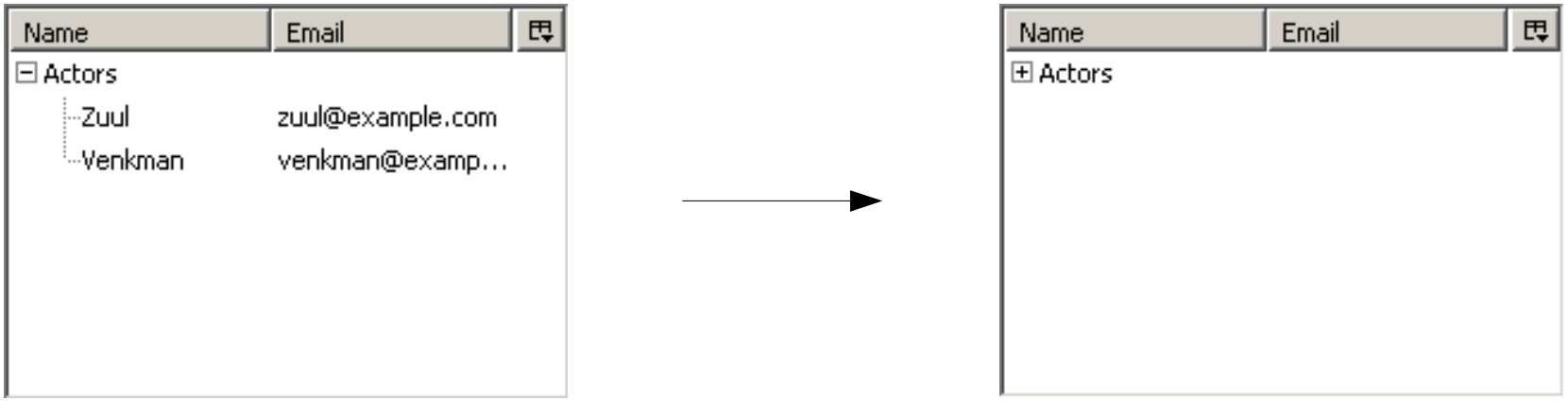
- Hierarchical tree
  - Multiple levels (0, 1, 2, ...)
  - Open/close subtrees
  - Lines connecting rows under same parent
  - Needed steps:
    - Make treeitem a “container” to allow open/close
    - Set primary=”true”-attribute on the first column (column where open/close and lines will be)

# Trees

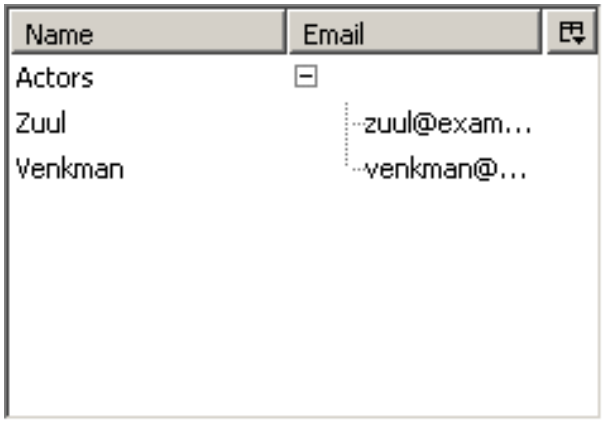
```
<tree flex="1">
  <treecols>
    <treecol id="Name" label="Name" primary="true" flex="1"/>
    <treecol id="Email" label="Email" flex="1"/>
  </treecols>
  <treechildren>
    <treeitem container="true" open="true">
      <treerow>  <treecell label="Actors"/>  </treerow>
      <treechildren>
        <treeitem>  <treerow>
            <treecell label="Zuul"/>
            <treecell label="zuul@example.com"/>
        </treerow>  </treeitem>
        <treeitem>  <treerow>
            <treecell label="Venkman"/>
            <treecell label="venkman@example.com"/>
        </treerow>  </treeitem>
</treechildren>  </treeitem>  </treechildren>  </tree>
```

# Trees

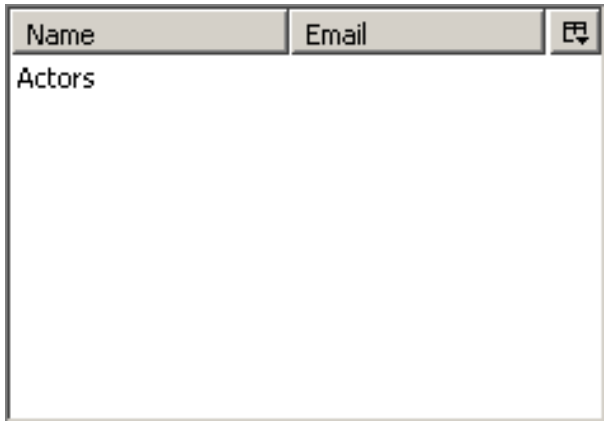
- Correct:




- Wrong primary column:




- Container attribute not set:



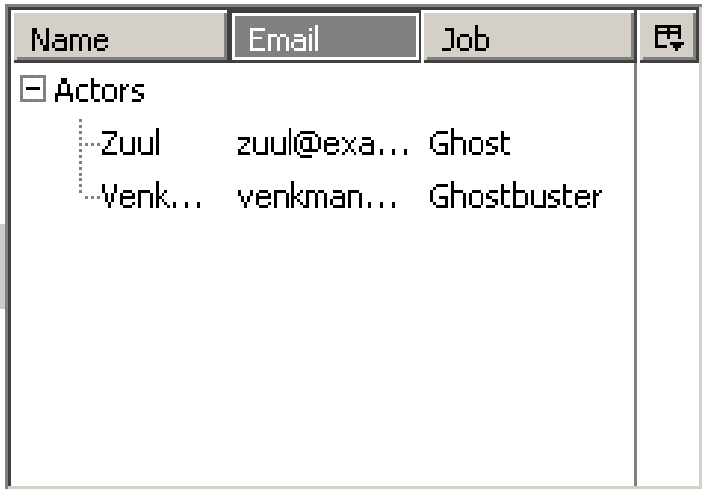
# Trees

- Hierarchical tree
  - One treerow + one treechildren per level
  - Nesting possible
  - Setting container-attribute important (containers without children possible)
  - Attribute open=”true” for container to be initially expanded

# Trees

- Advanced features
  - Reordering of columns:

```
<tree enableColumnDrag="true" />
```




  - Adjusting of column-size:
    <splitter> between treecols
    optionally with class="tree-splitter" to hide the notch
    - Again: setting minwidth and maxwidth allowed here
  - Saving state of columns between sessions: persist = "..."
    - width (Column widths), ordinal (position of column), hidden (Hidden-state of column)

# Trees

```
<tree enableColumnDrag="true" flex="1">
  <treecols>
    <treecol id="Name" label="Name" flex="1" primary="true"
     persist="width ordinal hidden"/>

    <splitter class="tree-splitter"/>

    <treecol id="Email" label="Email" flex="1"
     persist="width ordinal hidden"/>

    <splitter class="tree-splitter"/>

    <treecol id="Job" label="Job" flex="1"
     persist="width ordinal hidden"/>
  </treecols>

...
```

# Trees

- JavaScript and trees
  - Selected item for a seltype=”single”-tree
    tree-property “currentIndex”
    holds index (starting from 0) of current selected element
  - Selected items for a multi-select-tree
    use methods from object at “tree.view.selection”
    - Number of continous selected ranges: getRangeCount()
    - Get bounds for range r: getRangeAt(r, start, end)
    - Indexes start.value to end.value are selected
    - Checking for certain selection: isSelected(s)

# Trees

- JavaScript and trees
  - Setting selected items
    - rangeSelect(start, end, bool)  bool=add to curr select?
    - clearRange(start, end)
    - select(index)
    - toggleSelect(index)
    - invertSelection()
    - selectAll()

# Custom tree views

- Built-in content tree view just for few / simple data
- Custom view allows:
  - Large number of rows
  - Complex data to be shown (e.g. computed)
- Create an “empty tree”:

```
<tree id="squaretree" flex="1">
  <treecols>
    <treecol id="countcol" label="Count" flex="1"/>
    <treecol id="squarecol" label="Square" flex="1"/>
  </treecols>
  <treechildren/>
</tree>
```

# Custom tree views

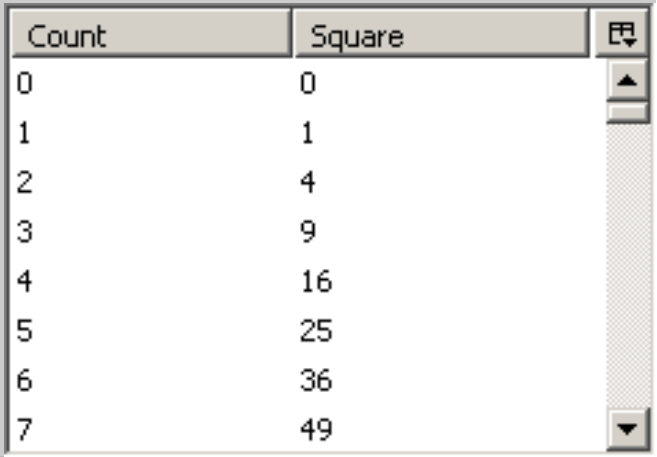
- Create object that implements “nsITreeView”-interface:

```
var treeView = {
    rowCount : 1000,
    getCellText : function(row, column){
      if (column == "countcol") return row;
      else return row*row;
    },
    setTree: function(treebox){ this.treebox = treebox; },
    isContainer: function(row){ return false; },
    isSeparator: function(row){ return false; },
    isSorted: function(row){ return false; },
    getLevel: function(row){ return 0; },
    getImageSrc: function(row,col){ return null; },
    getRowProperties: function(row,props){},
    getCellProperties: function(row,col,props){},
    getColumnProperties: function(colid,col,props){}
};
```

# Custom tree views

- Creating the window, applying the view:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window  xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
onload="setView();">
<script>
var treeView = {  [...]  };
function setView()
{
  document.getElementById('squaretree').view =
    treeView;
}
</script>
<tree id="squaretree" flex="1">
[...]
</tree>
</window>
```



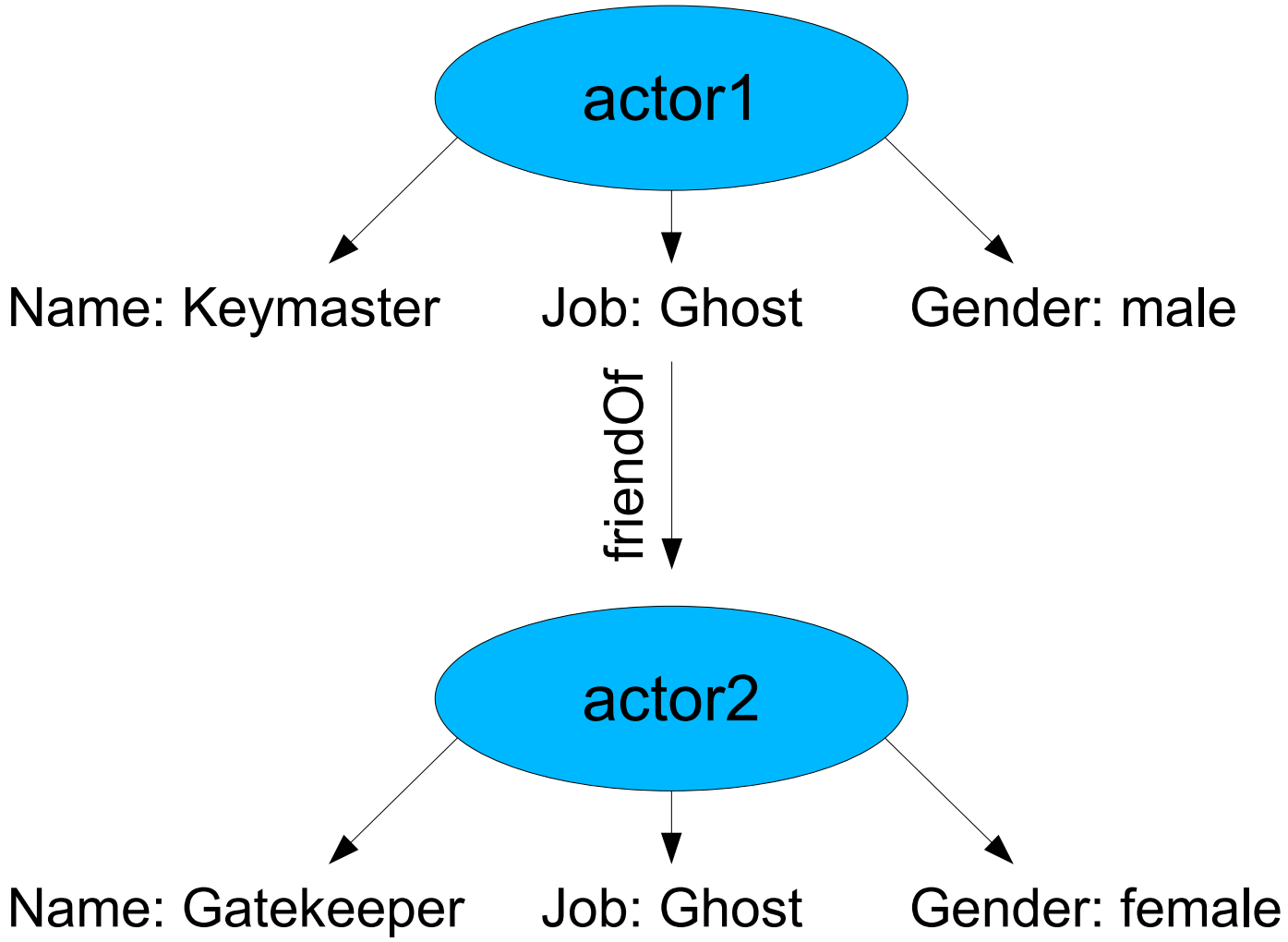
# Custom tree views

- The “nsITreeView”-interface:
  - Implement in JavaScript-object or using XPCOM-component
  - Almost 30 methods; For basic view only 10 methods needed
  - Hierarchical view:
    - also need to respond to isContainer(row), isContainerEmpty(row) and isContainerOpen(row)
    - View must keep track of open-states
    - Open/close triggers: toggleOpenClose(row)
    - Methods for hierarchy and linking: getLevel(row), hasNextSibling(row, afterIndex), getParentIndex(row)

# RDF

- RDF (Resource Description Framework)
- Format used to store e.g. bookmarks, history, emails
- Using existing RDF-sources or RDF stored in XML
  (XML is often used for storing – RDF is not by definition in XML)
- A set of resources represents e.g. a person
- RDF describes relations between resources:
  connection of nodes
- Nodes are “literals” (actual values)
  or “resources” (think of things like objects)
- Resources identified by URIs (unique)

# RDF


actor1
Name: Keymaster
Job: Ghost
Gender: male
friendOf
actor2
Name: Gatekeeper
Job: Ghost
Gender: female

# RDF

- Relationships called “triples” or “arcs”:
  actor1 → friendOf → actor2

  Parts of a triple called:
  subject → predicate → object / target
- Subject: always resource; object/target: resource or literal
- Subject, predicate and object described using URIs:
  - http://www.example.com/rdf/actor1
    - http://www.example.com/rdf/friendOf
      - http://www.example.com/rdf/actor2

# RDF

- Defining a type:

  http://www.example.com/rdf/actor1 →
  rdf:type → http://www.example.com/rdf/actor

- Reference to the RDF-namespace used:
  “rdf:type”
  here means
  “http://www.w3.org/1999/02/22-rdf-syntax-ns#type”

- Predicate “type” is defined in the RDF namespace already

# RDF

- Lists possible using “rdf:_1”, “rdf:_2”, ...

  http://www.example.com/rdf/allActors →
  rdf:_1 → http://www.example.com/rdf/actor1
  http://www.example.com/rdf/allActors →
  rdf:_2 → http://www.example.com/rdf/actor2

- List-types: “rdf:Seq” (ordered list), “rdf:Bag” (unordered list), “rdf:Alt” (list of alternatives)

  http://www.example.com/rdf/allActors →
  rdf:type → http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq

# RDF

- Storing RDF in XML: RDF/XML
  - RDF-tag as node of XML-document
  - Declare namespaces

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

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:actor="http://www.example.com/rdf/actor">

  <rdf:Description rdf:about="http://www.example.com/rdf/actor1"
   actor:name="Keymaster"
   actor:job="ghost"
   actor:gender="male"/>

</rdf:RDF>
```

# RDF

- “rdf:Description”: describes a triple
- “rdf:about”: specifies the subject of the triple
- Giving predicates, here from namespace “actor”
- Assigning target values

- Also possible to give separate triples for one subject:

```
<rdf:Description rdf:about="http://www.example.com/rdf/actor1"
 actor:name="Keymaster"/>
<rdf:Description rdf:about="http://www.example.com/rdf/actor1"
 actor:job="ghost"/>
<rdf:Description rdf:about="http://www.example.com/rdf/actor1"
 actor:gender="male"/>
```

# RDF

- Or use tags to specify predicates:

```
<rdf:Description rdf:about="http://www.example.com/rdf/actor1">
  <actor:name="Keymaster"/>
  <actor:job="ghost"/>
  <actor:gender="male"/>
</rdf>
```

- Or even mix:

```
<rdf:Description rdf:about="http://www.example.com/rdf/actor1"
 actor:name="Keymaster"
 actor:job="ghost">
  <actor:gender="male"/>
</rdf>
```

# RDF

- Resource-to-resource triples:

```
<rdf:Description rdf:about="http://www.example.com/rdf/actor1">
 actor:name="Keymaster">
 <actor:friendOf rdf:resource=http://www.example.com/rdf/actor2"/>
</rdf>
```

- Or nested:

```
<rdf:Description rdf:about="http://www.example.com/rdf/actor1">
 actor:name="Keymaster">
  <actor:friendOf>
    <rdf:Description
     rdf:about="http://www.example.com/rdf/actor2"
     actor:name="Gatekeeper"/>
  </actor:friendOf>
</rdf>
```

# RDF

- Lists in RDF/XML:

```
<rdf:Seq rdf:about="http://www.example.com/rdf/allActors">
  <rdf:_1 rdf:resource="http://www.example.com/rdf/actor1"/>
  <rdf:_2 rdf:resource="http://www.example.com/rdf/actor2"/>
</rdf:Seq>
```

- Or using automatic numbering of items:

```
<rdf:Seq rdf:about="http://www.example.com/rdf/allActors">
  <rdf:li rdf:resource="http://www.example.com/rdf/actor1"/>
  <rdf:li rdf:resource="http://www.example.com/rdf/actor2"/>
</rdf:Seq>
```

Note for using an RDF APIs:
Reading document translates rdf:li-tags to item-numbers

# Templates

- To create elements from datasources
  - Internal datasource: bookmarks, history, mails
  - In-memory datasource: Object of nsIRDFDataSource
  - RDF/XML-file
- Use: <template> ... </template>
- Place <template> in a container for all elements, e.g. a tree
- Elements in template repeated for each constructed element
- Repeat will start where: uri="rdf:*"

# Templates

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:actor="http://www.example.com/rdf/actor#">
 <rdf:Seq rdf:about="http://www.example.com/rdf/allActors">
  <rdf:li>
   <rdf:Description rdf:about="http://www.example.com/rdf/actor1"
    actor:name="Keymaster"
    actor:job="ghost"
    actor:gender="male"/>
   </rdf:li>
   <rdf:li>
    <rdf:Description rdf:about="http://www.example.com/rdf/actor2"
     actor:name="Gatekeeper"
     actor:job="ghost"
     actor:gender="female"/>
   </rdf:li>
  </rdf:Seq>
</rdf:RDF>
```

actors.rdf

# Templates

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window xmlns=
 "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
 <tree id="treefromrdf" flex="1" datasources="actors.rdf"
  ref="http://www.example.com/rdf/allActors">
   <treecols>
     <treecol id="Name" label="Name" primary="true" flex="1"/>
   </treecols>
   <template>
     <treechildren flex="1">
       <treeitem uri="rdf:*">
         <treerow>
    <treecell label="rdf:http://www.example.com/rdf/actor#name"/>
         </treerow>
       </treeitem>
     </treechildren>
  </template>
</tree>  </window>
```

actors.xul

# Content areas

- Displaying HTML or other XUL-files,
  embedded as one element into a XUL-window
- Simple element: <iframe>
  Supports basic features and is “lightweight”

```
<iframe id="content" src="http://www.mozilla.org" flex="1"/>
```

- Advanced element: <browser>
  Similar to <iframe> - but features page history, referers etc.

```
<browser id="content" src="http://www.mozilla.org" flex="1"/>
```

- Browser with tab-support: <tabbrowser>
  Widget that the Mozilla browser uses

```
<tabbrowser id="content" src="http://www.mozilla.org" flex="1"/>
```

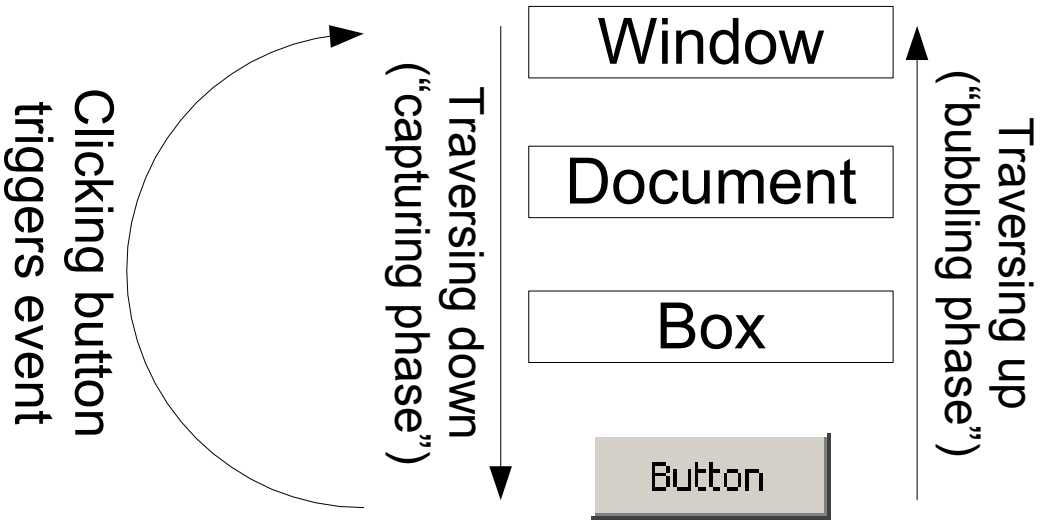
# Content areas

- Types of <browser>-elements
  - type=”...” NOT set
    Browser-content part of current window
    XUL-code in loaded content has access to the whole window
  - type=”content”
    Browser-content restricted to it's window
  - type=”content-primary”
    With multiple browsers one is primary
    Primary content accessible using “window.content”
- <tabbrowser> automatically sets “content” / “content-primary”
  “content-primary” is currently selected tab

# Event handling

DOM-events

- Examples: moving mouse or pressing a key
- XUL-elements can trigger certain events
- Event transfered as object
- Event traverses element-hierarchy
- Note: Some events have no “bubbling phase”



# Event handling

- Possible to attach listeners at each propagating step
  - Allows handling events for all sub-elements
- Possible to prevent event from propagating further

- Listening using element-attributes (e.g. command="...")
  - Easy to use, but only receives bubbling events
- Listening to events using addEventListener-method
  - Possible to listen at any phase
  - Multiple listeners per event/element

# Event handling

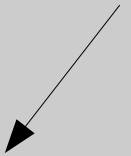
- Event received during bubbling, for all sub-elements
- Event-object-property “target”: where event occurred

```
<vbox oncommand="alert(event.target.tagName);">
  <button label="OK"/>
  <checkbox label="Show images"/>
</vbox>
```

- Listening in “capturing phase”, using DOM event handler

```
<button id="okbutton" label="OK"/>
<script>
  function buttonPressed(event)
  { alert('Button was pressed!'); }
  var button = document.getElementById("okbutton");
  button.addEventListener('command', buttonPressed, true);
</script>
```

Listen in “capturing phase”?



# Event handling

Event-object contains (extract):

- Constants:
  - CAPTURING_PHASE, BUBBLING_PHASE
- Properties:
  - eventPhase — Current phase of flow
  - boolean bubbles — Can this event bubble?
  - currentTarget — EventTarget who's EventListener is being processed
  - target — Original EventTarget for event

# Event handling

Event-object contains (extract):

- Methods:
  - stopPropagation()
    Propagation stops on current EventTarget

```
<hbox oncommand="alert('Box');">
  <button label="Button1" oncommand="alert('Button1');"/>
  <button label="Button2" oncommand="alert('Button2');"/>
</hbox>
```

- Clicking button1 results in: alert('Button1'); alert('Box');
  - Propagation not stopped
  - Processing in bubbling-phase (bottom-up)

# Event handling

Mouse-events:

- click, dblclick, mousedown, mouseup
- mouseover, mouseout
- mousemove

```
<script>
function mouseCoords(event)
{
  var coordLabel = document.getElementById("coords");
  coordLabel.value = event.clientX + ", " + event.clientY;
}
</script>

<box width="100" height="100" onmousemove="mouseCoords(event);"/>
<label id="coords"/>
```

# Event handling

Keyboard-shortcuts:

- Using accesskey="...", as already mentioned
- Keyboard-events:
  keypress, keydown, keyup

- Defining shortcuts for a window

```
<keyset>
  <key id="test-key" modifiers="control,shift" key="T"
   oncommand="alert('Test-key pressed');"/>
</keyset>
```

# Event handling

Keyboard-shortcuts:

- Modifiers:
  - shift, control, alt, control, accel (Win: “control”, Unix: “alt”), meta (Mac: “command key”; not on all platforms)
  - Adjusting keys: http://www.mozilla.org/unix/customizing.html
- Using defined key-IDs

```
<keyset>
  <key id="cut_cmd" modifiers="accel" key="X"/>
</keyset>
[...]
  <menuitem label="Cut" accesskey="t" key="cut_cmd"/>
```

# Commands

- Aim: Separate application tasks and user interface
- Design goal: Strict separation, plan functionality before start
- Central (or even external) definition of commands
- Defined commands can be disabled or otherwise modified

```
<command id="cmd_greeting" oncommand="alert('Hello user!');"/>
<button label="Greeting" command="cmd_greeting"/>
```

- Events and commands are different (though they interoperate)
- Here: Button generates DOM 2 event of type “command”

# XUL with PHP?

- Sending the correct mime-header for XUL:

```
<?php
header( "Content-type: application/vnd.mozilla.xul+xml" );
?>
```

- Generating RDF-sources (XML-data) using PHP
- RDF-output-drivers for some PEAR-packages in the works (more hopefully at PHP Conference Frankfurt 2005)
- PEAR-package to help create XUL-code: http://pear.php.net/package/XML_XUL

  (Package and examples by Stephan Schmidt)

# XUL with PHP?

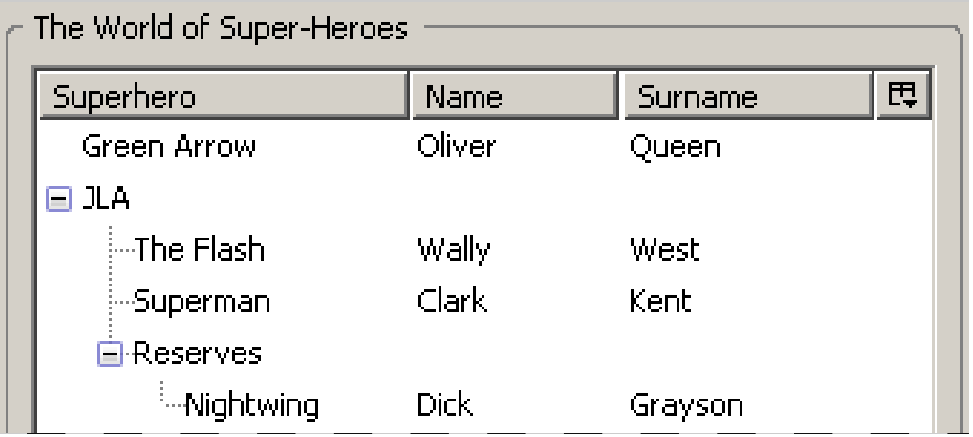
```
<?php
require_once 'XML/XUL.php';

$doc = &XML_XUL::createDocument( );

$doc->addStylesheet('chrome://global/skin/');

$win = &$doc->createElement('Window', array('title'=> 'Example for
PEAR::XML_XUL'));
$doc->addRoot($win);

$gbox =  &$doc->createElement('Groupbox',
array('orient'=>'horizontal', 'height' => 500));
$win->appendChild($gbox);

$gbox->setCaption('The World of Super-Heroes');
```

# XUL with PHP?

```
$tree = &$doc->createElement( 'Tree', array( 'flex' => 1 ) );
$tree->setColumns( 3,
                        array(
                                'id'  => 'hero',
                                'label' => 'Superhero',
                                'flex'  => 2,
                                'primary' => 'true'
                              ),
                        array(
                                'id'  => 'name',
                                'label' => 'Name',
                                'flex'  => 1
                              ),
                        array(
                                'id'  => 'surname',
                                'label' => 'Surname',
                                'flex'  => 1
                              )
                  );
```

# XUL with PHP?

```
$tree->addItem(array( 'Green Arrow', 'Oliver', 'Queen' ));

$jla = &$tree->addItem(array( 'JLA' ));

$jla->addItem(array( 'The Flash', 'Wally', 'West' ));
$jla->addItem(array( 'Superman', 'Clark', 'Kent' ));

$reserves = &$jla->addItem(array( 'Reserves' ));
$reserves->addItem(array( 'Nightwing', 'Dick', 'Grayson' ));

$gbox->appendChild( $tree );

/* ... shortened ... */

$doc->send();
}
?>
```



# Submitting a form

- Including form using HTML-namespace works (but nobody would do that!)
- Sending “form” using XUL with XML-RPC or SOAP (requires elevated rights; see articles in resources-list)
- Simple request:

```
// create object
req = new XMLHttpRequest();

req.open('GET', 'http://www.example.com/myscript.php', false);
// last param: false=wait for request to finish (synchronous)

req.send(null);  // param: used for body of a POST-request
// req.status:       http-response-code (200=success)
// req.responseText: text of response
```

# Submitting a form

- Asynchronous request:

```
req = new XMLHttpRequest();
req.open('GET', 'http://www.example.com/myscript.php', true);
req.onreadystatechange = readyStateChangeHandler;

req.setRequestHeader('User-Agent', 'MyAgent'); // optional
req.send(null);

function readyStateChangeHandler () {
  if (req.readyState == 4) { // 4=request completed
    if(req.status == 200) {
      // response available
    } else {
      // error during request
    }
  }
};
```

# Submitting a form

- Performing GET-request with variables:
  Combine URL and variables before request

```
[...]
var txt = "?1";
for(var i in this.parms) {
  txt = txt+'&'+this.parms[i].name+'='+this.parms[i].value;
}
//Two options here, only uncomment one of these
//GET REQUEST
httpRequest.open("GET", targetURL+txt, false, null, null);
[...]
```

(taken from complete example by Harry Fücks, available at:
http://www.phppatterns.com/index.php/article/articleview/82/1/2/ )

- Consider using a wrapper-library (e.g. jslib):
  http://jslib.mozdev.org/libraries/network/http.html

# Future of XUL

- Feature-rich and extensible
- Localization and customization
- Platform-independent
- Increasing number of Mozilla-platform-installations
- Large community
- Free
- XUL-like implementations e.g. for Java

- Comparison to XAML (Microsoft) ?

# Resources

- XUL Programmer's Reference
  http://www.mozilla.org/xpfe/xulref/
- Creating Applications with Mozilla (complete O'Reilly book)
  http://books.mozdev.org/
- Tutorials @ XULPlanet
  http://www.xulplanet.com/tutorials/
- http://developer.mozilla.org/en/docs/XUL_Tutorial
- Rapid Application Development with Mozilla
  2003, Prentice Hall, by Aussie Nigel McFarlane
- Neil's Place – blog on XUL, by Neil Deakin
  http://www.xulplanet.com/ndeakin/
- XUL widget cheatsheet
  http://www.mozilla.org/docs/xul/xulnotes/xulnote_cheatsheet.html

# Resources

- Good overview of widget features and variants
  http://www.hevanet.com/acorbin/xul/top.xul
- XUL and XML-RPC
  http://www.phppatterns.com/index.php/article/view/86/1/2
- Mozilla and SOAP
  http://www.oreillynet.com/pub/a/javascript/synd/2002/08/30/mozillasoapapi.html
  http://docs.mandragor.org/files/Misc/Mozilla_applications_en/mozilla-chp-12-sect-9.html
- Mozilla Amazon Browser
  http://www.infodraft.com/~faser/mab/
  http://www.oreillynet.com/pub/a/mozilla/2003/05/02/casestudy2.html
- http://en.wikipedia.org/wiki/XUL
- http://en.wikipedia.org/wiki/Comparison_of_user_interface_markup_languages

# Thank you!

Up-to-date slides available at:

http://talks.speedpartner.de/

Questions?
neufeind (at) speedpartner.de