
[Home](#) [Experiments](#) [Applescripts](#) [Contact](#)

Experiments

Every now and then, an article or an interesting discussion brings up an idea that could improve the way we use our computers. Whenever I find the time to try out one of those ideas, and the results are interesting, the demos and descriptions can be found here.

Bringing The Humane Interface to OpenOffice.org

June 2009

In late 2008, OpenOffice.org announced the project "Renaissance" with the stated goal of giving the office suite's user interface a complete overhaul. To collect some ideas, there has been a [public call for proposals](#) [↗](#), which I couldn't resist. You can read [my proposal "Habituating Interaction"](#) [↗](#) at the OpenOffice.org wiki.

Claim Back Your Browser

December 2008

Bookmarklets, or favelets, are old news. But even today, many people never heard of them, and the power and versatility of bookmarklets is frequently underestimated. They are a convenient way to let you, the user, do anything you like with the contents of your browser, without having to download extensions or plugins. Text arranged in an illegible manner? Side column full of distracting animations? Simply change it. Want to use some info on the page in some other web service (like google mail)? Just click "send as google mail". It's your browser, so you should be in control. There's no need to use a particular browser or install browser extensions. Bookmarklets work in your favourite browser, and are a non-proprietary way to handle these tasks perfectly.

From a technical perspective, bookmarklets are little snippets of JavaScript, neatly packaged into an URI starting with "javascript:...", that can be stored in the bookmarks bar of your browser (e.g. by drag and drop). Clicking the bookmarklet executes the JavaScript code in the current Web page.

This makes it possible to re-gain control over what's going on in your browser window. Want to remove that irritating ad? Just go ahead and [delete](#) it. Want to make that column wider or narrower? Simply [resize](#) it. And so on [and so on](#) [↗](#).

Here are a few example bookmarklets. Drag them to your bookmarks bar to use them. If you're using Safari, you can also access them through a CMD-**<Number>** shortcut, where **<Number>** corresponds to their position on the bookmarks bar.

- [Delete](#) lets you remove elements in the page (paragraphs, columns, pictures, ...) by clicking on them. Just don't click directly at flash elements, as they're tricky beasts and will notice the click anyway. To remove flash blocks, you need to remove their surrounding block. You'll usually find one by moving your mouse around a little around their borders.
- [Resize](#) allows you to widen or narrow any block on the page. Useful for pages that seem to be made for screens with a width of 10000 pixels, as well as for those that have mile-long text columns with only 2 words per line.
- [Map](#) opens Google map with the currently selected address (in a new window; it lets you edit the address before submitting the query, because I found that you often deal with partial addresses).
- [Images](#) lists Google Images results found with the selected text (in a new window; it asks you for search terms if nothing has been selected).
- [Google Preview](#) shows off what bookmarklets can do. Click this bookmarklet on a google search results page to get mouse-over previews for the search results.
- [Back](#) and [Forward](#) decrement or increment the last number in the URL. Useful for flipping pages on some sites (not only on icanhascheezburger).
- [Open Link](#) opens the currently selected link. This is particularly useful in Safari: Search for a link text, like "download", and you'll notice that the current search occurrence is selected. When you invoke the bookmarklet, the highlighted link will be followed. Example: CMD-F, "downl", CMD-1.

Apple Inc. v. Smith Interface Techs., LLC

OpenOffice.org: Function Assistance in Cells

IPR2024-01083 | Smith EX2012

Page 1 of 9

March 2008

Here's another series of live mockups to support a discussion in the OpenOffice.org User Experience discussion group which dealt with the [question of how to best provide assistance when entering formulae into spreadsheet cells](#). The mockups do not necessarily represent my idea of how to approach the problem, but were meant to support the discussion in making the suggested behaviors tangible and easier to evaluate.

[Read more...](#)

Enso: To make simple things simple...

...and complex things possible.

February 2008

Humanized recently [proposed a new design](#) for their launcher *Enso*. Below, I'll discuss its advantages and drawbacks, and suggest a new sub-application, called *Scratchpad*, that would solve the problems of the current approach. If you're the visible type, you can start by watching a quick tour ([part one](#), [part two](#)) or try out the [Enso Interface Demo](#) yourself.

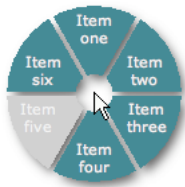
The idea of an Enso scratchpad was first brought up by Braydon Fuller in a [discussion at humanized.com](#).

[Read more...](#)

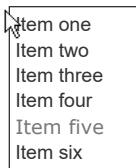
Pie menus on the Web

January 2008

A pie menu, or radial menu, is a kind of pop-up menu whose items are arranged in a circle around the mouse cursor:



Here are the same items arranged in a linear menu:



Pie menus can be used in web apps, like for example in the famous [Songza](#) or in webmail applications, where you can click on an email in your inbox to get a menu with possible actions (e.g. open, reply, mark as spam, delete). Commands that you would put in a traditional context menu.

The advantage of pie menus over traditional linear menus is that all its items are very easy to target with the mouse cursor. That's mainly because of two reasons:

1. We are much better in moving the mouse in a particular direction than in moving it a particular distance.
2. Items closer to the mouse cursor are easier to hit.

Try it! Click anywhere into this text paragraph to activate the menu and select an option. Now try the following: Before invoking the menu, think of the sector you want to click (e.g. top, lower right, etc.). Then quickly invoke the menu and click this sector immediately. You will find that you can hit any sector very quickly, almost without looking and with surprising precision. Soon there won't be any hesitation between summoning the menu and selecting an option. It becomes one single gesture.

Part of this effect is explained by [Fitts's Law](#): The time one needs to click on a target is a function of the target's size and its distance. Therefore, no two items in a traditional linear menu are equally easy to click. That's why item order is critical in linear menus. It is much less of an issue for pie menus.

Another effect of pie menus is that their use becomes habitual. That means that after having selected an item a couple of times, we do not need to read its label any more, we even almost don't have to look at it any more. For example, there's an on-screen object, and clicking it invokes a pie menu where the south item means "mark as read". To mark that item as read, you click it, move the mouse a bit to the south, and click again. Soon you will become very quick at performing these steps, and you will stop paying attention

to the menu selection as you make it. Your muscle memory will do it for you. The invocation of the pie menu and the selection of the item have been fused into one single, habituated, gesture. **Habituation** ² is also the reason why it is so important for the same menu to always have the same item ordering. This rule applies to linear menus, too. Nothing is more frustrating than clicking on a spot you've clicked a hundred times before, only realizing a split-second later that this time, the item didn't read "duplicate" but "delete".

Pie menus were first devised by **Don Hopkins** ³ et al. back in the 80s. On his site, you will also find much original research on pie menus.

Behavior and layout of pie menus

Pie menus are invoked in just the same way you would invoke an ordinary menu. For example by left- or right-clicking an item on the screen. In fact, context menus are a perfect use for a pie menu layout. It is crucial that the pie menu centers exactly around the current mouse cursor position, or it would become less efficient to use, even less so than a traditional linear menu.

If we want to make our menu keyboard-invokeable, we should also make it keyboard-controllable, e.g. with the cursor keys (up key selects north item, down left keys select southwest item, etc.). Invocation by keyboard doesn't make sense for solely mouse-operated pie menus. Because the user has to take hold of the mouse, they can't habituate the menu. In this example, we'll focus on solely mouse-operated pie menus.

Pie menus are dismissed without making a selection when the user clicks in their middle, or outside of them (same as with traditional menus), or by moving the mouse more than a certain distance away from the menu. This diminishes the risk of accidentally dismissing the menu when overshooting. I suggest this buffer distance be about equal to the menu's diameter.

In the middle of the menu, there will be a circular area, acting as a dead zone. Nothing gets selected when the user clicks in this area. Its purpose is to easily dismiss the menu and to prevent accidental selection of the wrong item. This zone should be larger if the users are not proficient mouse users, if the use situation may be stressful, or if the users may be in a hurry. But at the same time, we shouldn't make the dead zone too large, or we are wasting the advantage of having the menu items near the mouse cursor.

The respective click-areas (hotspots) of the items are sector- or wedge-shaped (with a bit at their inner tip missing because of the dead zone). They form a ring around the mouse cursor. The wider we make this ring, the easier it is to hit an item. Even if the respective items present themselves like rectangular icons or other shapes, the underlying hotspots should always be sectors or wedges, or we lose many advantages of pie menus.

Menu appearance

Like a normal context menu or pull-down menu, a pie menu should look like it temporarily floats over the original screen content. This is even more critical as most users are still unfamiliar with pie menus and otherwise might think it only was some odd circular image that for some reason has replaced the clicked-on object. So, in a way, floating suggests menu-ness. We can support this by animating the menu consistently when opening and dismissing it, to clearly show where it comes from. We elevate it from the page with a drop-shadow and we can further suggest its temporary nature by making it partially translucent.

Item appearance

We best make the presentation show how to use the menu, particularly as we want to quickly get across the idea of how to use it. As the hotspots are sector-shaped, we also should make the items sector-shaped. This is sometimes more difficult to achieve technically than working with ordinary rectangular icons, but worth the trouble. Additionally, the sectors should best have little gaps between them. This tells the user that the items are individually selectable entities. This is particularly important if you give them pseudo-physical behavior, similar to buttons. If you want the sectors perceived as being "touchable", they should look loose and "moveable", and not like if they were fused to each other. In the real world, you've probably already come across buttons that didn't look like buttons at all. On some microwaves, and on heavy machinery, buttons sometimes only appear as rounded rectangles drawn on smooth surfaces. That's because they have to be easy to clean. But avoid this on the Web. Good UI elements suggest (or **afford** ⁴) how to use them.

Some pie menus use text, some use icons. As a personal side note, I'm strongly biased towards text, because you can't just "read" icons you've never seen before. That is, most unfamiliar icons have to be explained before they can be used. And, ironically, that explanation is given in the form of written text (e.g. in a tooltip). Often, it's better to just use text in the first place. It saves one step of interpretation and does not burden the user with an additional symbol to memorize.

Inactive, highlighted and selected states

Inactive menu items should not be removed from the menu, but rendered inactive (e.g. "grayed out"), to support the users' habit-building ability.

Most users will not be familiar with pie menus. That's a serious problem. In order to suggest that the sectors can be clicked on, in addition to the sector's look we are using a mouseover highlighting effect that shifts and enlarges the sector slightly when the user moves their mouse cursor over that sector.

When the user clicks on a sector, it's a good idea to make the selected sector appear "pushed down", similar to a button. That completes the pseudo-physical behavior of the menu. [I'm skipping this step to keep the example lean and mean.]

Item arrangement

Fortunately, unlike with linear menus, we can't really make bad ordering mistakes here, as all items are about equally easy to target. The items straight north, south, west and east are a little easier to hit than the diagonal ones, but that's about all. When arranging them, we should keep in mind that most often, the items have a meaningful relationship to each other. Here is in fact another advantage of pie menus that linear menus don't have: Thanks to the spatial arrangement, we can provide the user with **natural mappings** as additional cues. For example, we could position a "go back" item to the west, a "go forward" item to the east, a "download" item down south, and so on.

Where on the web can pie menus be used?

[This section is still fairly incomplete]

Applications for pie menus are rare, but rewarding. One shouldn't be tempted to replace ordinary links or navigation with pie menus. They still have one very basic shortcoming in common with regular pop-up menus: Lack of visibility.

Pie menus mustn't be used where they cannot be placed around the mouse cursor.

Don't use pie menus if the set of menu items presented can change. This forfeits habituation. If menu items can change, use an always-open, non-pop-up, linear menu (while selection in pie menus beats linear selection by lengths, finding a new item is far easier in a linear list or menu than in a pie menu).

Don't use pie menus near the border where a part of the menu may be cut off. Shifting the menu inwards to be fully visible is not a solution, because the habituated ("blind") gesture will now miss the desired item and maybe even select a wrong one (The mouse pointer would not be centered in the pie menu). Moving the mouse pointer accordingly (if at all possible) is not a solution, because it disorients and thus slows down the user (and most likely even distracts the user from the task they wanted to achieve).

Possible applications are not on ordinary not-particularly-interactive web sites, but rather on mini applications like the famous [Songza](#) or a web mail application. (As a side note, I find Songza's interface very well thought-out, but the way the pie menu works doesn't quite agree with me in detail, particularly the sub-menus to the left and to the right, that they're are relatively hard to hit and interfere with dismissing the menu.)

A comprehensible implementation of pie menus

What we need:

- some images
- HTML
- JavaScript

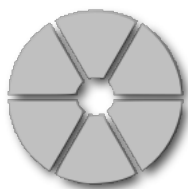
Some assembly required.

Although there are several different approaches for rendering shapes in the browser, I found it easiest to use images for the pie menu. Some other relevant techniques are [SVG](#), [building the image out of tiny DIV elements](#) or [generating bitmaps using JavaScript](#). And, of course, there are also approaches using Flash and [server side](#) techniques. But all of those quickly become unwieldy, and I wanted my example to be as clear as possible.

So, say we want a pie menu with six sectors. In fact, I should have made it eight sectors, because with six, I'm wasting the straight west and east directions. Anyway, we first need an image of the inactive menu. This is a GIF with a transparent background. Note that I left some room on each side. This is because I made all images to have the same dimensions, so there will be no hassle when positioning the individual elements later.



Out of that image, we can build a drop shadow using our favorite graphics manipulation software. This is a PNG image with an alpha channel. In Internet Explorer 6, its background will appear opaque (not transparent). There is [a way to make IE6 display PNG alpha channels](#) ², but I'll leave this out for now to keep things simple.



Now we make a copy of our first "inactive" GIF image, change its color, and make six images for each of the sectors. The background, again, is transparent.



, ...

Next we make six images for the mouseover highlighting effect, with transparent background. (If you want to keep the number of images low, you could also use [a technique to put all pie sectors in one single image](#) ³.)



, ...

And last but not least, we make another GIF that's completely transparent and thus invisible. I made this one the same size too. It will accommodate a clickmap of the sectors.

(invisible image)

Stacking the elements

This is the stacking order of the menu's elements, from back to front:

1. Drop shadow
2. Inactive menu (I didn't combine this with the shadow image because of IE6's problems with PNGs with alpha channel)
3. Sectors one to six (for inactive sectors just hide their corresponding image here)
4. Active sectors one to six (all hidden)
5. An unordered list containing the sectors' labels or icons (I prefer textual labels)
6. Transparent covering image (with clickmap)

The resulting HTML looks like this:

```
<div id="radialMenu">
<div id="radialViewContainer">
```

```


















<ul>
  <li id="radialText1">Item<br/>one</li>
  <li id="radialText2">Item<br/>two</li>
  <li id="radialText3">Item<br/>three</li>
  <li id="radialText4">Item<br/>four</li>
  <li id="radialText5">Item<br/>five</li>
  <li id="radialText6">Item<br/>six</li>
</ul>
</div> <!--end: id="radialViewContainer"-->



<map name="radialMenuMap">
  <area shape="poly" coords="93,88,62,32,99,18,137,33,107,88" href="javascript:radialClick(1)" alt="" onmouseover="highlight(1)"
  <area shape="poly" coords="107,88,137,33,171,62,176,100,113,100" href="javascript:radialClick(2)" alt="" onmouseover="highlight(2)"
  <area shape="poly" coords="113,100,176,100,167,140,138,166,107,111" href="javascript:radialClick(3)" alt="" onmouseover="highlight(3)"
  <area shape="poly" coords="107,111,138,166,100,177,63,166,94,111" href="javascript:radialClick(4)" alt="" onmouseover="highlight(4)"
  <area shape="poly" coords="94,111,63,166,31,140,24,100,86,100" href="javascript:radialClick(5)" alt="" onmouseover="highlight(5)"
  <area shape="poly" coords="86,100,24,100,30,60,62,32,93,88" href="javascript:radialClick(6)" alt="" onmouseover="highlight(6)"
</map>
</div> <!--end: id="radialMenu" -->

```

Of course, I shouldn't really have used HTML event attributes and that many IDs: One should rather write some unobtrusive JavaScript that gathers all the information from the UL element, generates the necessary menu elements dynamically and calculates the hotspot coordinates itself. Or, instead of using hotspot coordinates, the JavaScript could also just use the angle of the current mouse position relative to the pie menu's center. But I like to present the HTML markup like I do because I think that its inner workings become more apparent that way.

This is the CSS:

```

#radialMenu
{
  position:absolute;
  width:200px;
  height:200px;
  opacity:0.9;
}
#radialMenu #radialViewContainer
{
  position:absolute;
  top:0px;
  left:0px;
  width:200px;
  height:200px;
}
#radialMenu img
{
  border:none;
  position:absolute;
  top:0px;
  left:0px;
  width:100%;
  height:100%;
}
#radialMenu img.active
{
  visibility:hidden;
}
#radialMenu ul
{
  margin:0;
  padding:0;
  position:absolute;
  top:0px;
  left:0px;
  list-style-type:none;
  color:#EEEEEE;
}

```

```

        font-size:8pt;
    }
    #radialMenu li
    {
        position:absolute;
        text-align:center;
        white-space:nowrap;
    }
    #radialMenu li.active
    {
        color:#FFFFFF;
    }
    #radialMenu #radialText1
    {
        top:39px;
        left:82px;
    }
    #radialMenu #radialText2
    {
        top:70px;
        left:120px;
    }
    /* ... */
    #radialMenu .active#radialText1
    {
        top:36px; /* slightly shifted outwards */
    }
    #radialMenu .active#radialText2
    {
        top:69px; /* slightly shifted outwards */
        left:122px;
    }
    /* ... */

```

I did not include the resizing steps from the animation in here. There's not much to the animation. Every image has its width and height given as 100% of its container, so I simply change their container's size to scale them.

Here is some JavaScript that controls the menu's behavior. (Sans animation. You can see the full code when you view this page's source).

```

function showRadial(evt)
{
    var p = mousePos(evt);
    var rm = document.getElementById("radialMenu");
    rm.style.left=(p.x-100)+"px";
    rm.style.top=(p.y-100)+"px";
    rm.centerCoords = p;
    rm.style.display="block";
    addEvent(document, "mouseup", hideRadial);
    addEvent(document, "mousemove", checkMouseToRadialDistance);
    cancelEvent(evt);
}
function hideRadial()
{
    document.getElementById("radialMenu").style.display="none";
    removeEvent(document, "mouseup", hideRadial);
    removeEvent(document, "mousemove", checkMouseToRadialDistance);
}
function highlight(nr)
{
    var ra = document.getElementById('radialActive'+nr);
    var rt = document.getElementById('radialText'+nr);
    if (!ra || !rt) return;
    ra.style.visibility="visible";
    rt.className="active";
}
function unhighlight(nr)
{
    var ra = document.getElementById('radialActive'+nr);
    var rt = document.getElementById('radialText'+nr);
    if (!ra || !rt) return;
    ra.style.visibility="hidden";
    rt.className="";
}
function radialClick(sectorNumber)
{
    showOverlayMessage(sectorNumber);
}
function checkMouseToRadialDistance(evt)
{
    var rm = document.getElementById("radialMenu");
    var pos = mousePos(evt);
    if (distance(pos.x,pos.y,rm.centerCoords.x,rm.centerCoords.y) > 200)
    {
        hideRadial();
    }
}

```

```

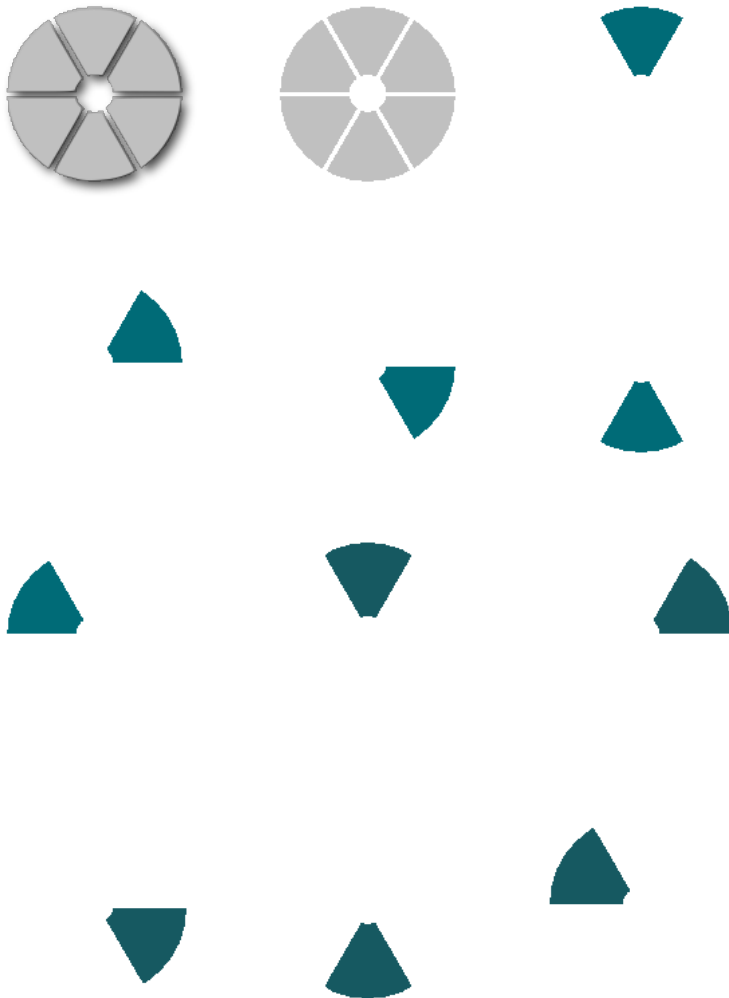
    }
}
// Some functions left out for brevity. Most of them can be found on the web:
// - distance calculates the distance between two points
// - addEvent adds an event handler
// - removeEvent removes an event handler
// - mousePos gets the current mouse position
// - showOverlayMessage shows a transparent message

```

And this is the result:

[Click here to activate the pie menu]

I hope that you enjoyed reading this article as much as I enjoyed writing it. If you've got comments or are interested in more details, just [use the contact form to get in touch with me](#).



- Item one
- Item two
- Item three
- Item four
- Item five
- Item six

OpenOffice.org: Modeless Page Arrangements

November 2007

I participated in an interesting [mailing list discussion](#) about introducing a new page view mode for the users who want to view two pages next to each other while editing. To be frank, I'm not a big fan of [modal user interfaces](#). I am deeply convinced that there must be a solution that satisfies most users while at the same time reducing the number of modes.

I made some [OpenOffice Interface Demos](#) that you can try out in your web browser. The only functional elements are the scroll area and some controls at the bottom of the window. My personal favorite is [Demo number 2](#), and if it didn't have that "facing pages" checkbox, there wouldn't be any view modes whatsoever!

[Read more...](#)



Except where otherwise noted, text, images and other media on this site are licensed under a [Creative Commons Attribution 2.0 License](#).

Except where otherwise noted, all source code is licensed under a FreeBSD (2-clause BSD) license. View page source for details.

[Andreas Schuderer](#)

Last updated: Saturday, 15-Jan-2011 21:19:49 CET