XWT – A Very Brief Introduction

by Andrew Kohlsmith <akohlsmith-kwpm@benshaw.com>
20030115

<Heading name="What is XWT?"/>

XWT stands for XML Windowing Toolkit. The idea behind XWT belongs to Adam Megacz, who started the project. It is essentially a Java engine which interprets XML-based widget files and presents them to a user. Unlike X11, everything is done on the client side. This means that button-pushes, slider controls, text entry – it all executes as fast as any native application. The widget set is also themeable, meaning that the user doesn't have to be presented with a widget set that they are not used to.

The basic functional block in XWT is the box. The top-level surface is your root box – everything is a descendant of that surface. Buttons are just boxes. Edit areas are just boxes. Sliders, icons, text... all boxes. The layout and operation of boxes in general is very heavily based on Donald Knuth's TeX.

As stated, the XWT engine is written in Java, but there are native binaries for Win32 and Linux. The engine is known to run on Win32, Linux and Mac OS/X. Other operating systems should work fine so long as there is a working Java 1.2 (1.4 preferred) runtime environment available.

All boxes have a number of properties (colour, size, text, alignment, spacing, etc.) which determine how the box looks and is positioned. You can create your own properties to handle just about anything you need in a widget. By assigning a function to an property, you create a trap – code which is executed when the property is accessed or changed. It is through the writing of functions that XWT applications take form. Applications can interact with the world thorough any XML-RPC or SOAP server. I use `Frontier::RPC2` with Perl but there is no reason why you can't access C++ or VB servers on Windows, Ruby servers, BASH servers... So long as it implements standard XML-RPC or SOAP, XWT should be able to talk to it.

All XWT functions are written in a stripped-down JavaScript which the XWT engine interprets at runtime. The implementation is a more or less complete ECMA-262 revision 3 (ECMAScript is what JavaScript's official name is) core, but without any of the OO bits. This isn't nearly as restrictive as it sounds – XWT's primary goal is to make and operate the UI – all of the complex business logic should be handled by your servers.

The engine runs the XWT application in a very restrictive sandbox. The only access methods are through XML-RPC or SOAP, or through specific Open or Save dialog boxes. All root windows are scarred with a red band and a lock symbol to prevent malicious programmers from creating fake login windows.

It is very easy to take people versed in HTML and JavaScript and give them some basic training on XWT and "pawn off" the actual presentation logic so that you can concentrate on getting the business end of things up and running.

<AudienceResponse type="Horror"/>

Yes, I mentioned JavaScript – the bane of the Internet. I just about wrote off XWT when I realized that XWT is centered around JavaScript, and that I' d have to learn JavaScript. Having worked with it for just about a year now, I have to say that selecting JavaScript was an excellent move on Adam' s part. It' s not a tacked-on afterthought like how JavaScript is under (D)HTML – it' s a core part of XWT and is complex enough to handle practically anything a widget will need to do. It' s simple and straightforward enough that you can take practically anybody who' s done web development and give them some basic training on XWT and they' ll find their way.

<Heading name="Sample XWT Code"/>

A sample XWT script:

```
<xwt>
    Any text can go here because it's not inside the template tag
    so open-ended text is possible for documentation and so on.

    <import name="org.xwt.stuff.*"/>

    <static>
        // code here will be executed only once
    </static>

    <template orient="vertical">
        <checkbox/>
        <box>
            <scrollbar/>
        </box>
    </template>

</xwt>
```

There you have a very basic dialog box. A checkbox beside a box with a scrollbar. How about something a little meatier?

```
__KeyPressed = function(k) {
    if (k == "escape")
            $cancel.click = true;
    else if (k == "enter")
            $ok.click = true;
    else
            arguments.cascade(k);
    }
```

That is a read trap on the `KeyPressed` property of this particular box. You can see that if someone pressed the escape key I in turn simulate clicking on the cancel button. Or if they hit the enter key I click on the ok button. And if any other key was pressed, I let any boxes I' ve created within my widget (including other widgets inside mine) see the keypress. (it' s known as cascading the trap, more on this later.)

Creating widgets is easy. This is the complete code for my status line widget:

```
<xwt>
    <import name="xwt.standard.*"/>

    <template vshrink="true">

        __text = function(t) {
            if (arguments.length == 0) return $statusline.text;
            $statusline.text = t;
        }

        <bevel depth="down" thickness="1">
            <box width="32"/>
            <box id="statusline" vpad="3" hpad="3" font="Helvetica11"/>
            <box width="3"/>
            <vbox shrink="true" align="bottomright">
                <box height="1" id="statusicon"/>
            </vbox>
            <box width="1"/>
        </bevel>
    </template>
</xwt>
```

```
<import name="path.*"/>
```

This implies a search path for widgets within the XWT file. When I ask for `<bevel>`, for instance, XWT will first search the current directory, then it will look through any imports it is given.

```
<template vshrink="true">
...
</template>
```

Any bare text in a `<template>` tag is interpreted as JavaScript. I am also specifying that I want the widgets within this block to take up as little vertical space as possible.

```
__text = function(t) {
         if (arguments.length == 0) return $statusline.text;
         $statusline.text = t;
         }
```

This is a read trap on my text property. If there are no arguments then whatever is accessing my text property wants my widget' s text, so I give it to them. Otherwise I set the status line text to whatever is specified. I have to specify a trap like this because anyone putting my statusline widget in their own application cannot access any boxes inside my widget without me providing properties to access them. This is important, as it tends to enforce OO concepts – that widgets handle their own internal processes and that any interaction is done through specifically-written properties and traps.

```
<bevel depth="down" thickness="1">
```

Here I want to create a “down” or sunken-looking bevel. Of course the bevel is just another XWT widget, much like this statusline widget I' m creating.

```
<box width="32"/>
<box id="statusline" vpad="3" hpad="3" font="Helvetica11"/>
<box width="3"/>
<vbox shrink="true" align="bottomright">
    <box height="1" id="statusicon"/>
</vbox>
<box width="1"/>
```

I am creating a number of boxes which are children of the bevel – i.e. The are “inside” of the bevel widget. They will all appear sunken down compared to the main window, which was the intention. Since the default box placement is horizontal, then vertical, these boxes will appear side-by-side. The first box is 32 pixels wide (to “skip over” the root-box scar), the next is the actual box that we put the status message in, and the third one is just for some padding between the text and the icon box.

The icon box is inside a vertical box (vbox, again just another XWT file) so that it can be aligned with the bottom of the widget I' m designing. ("bottomright" specifies bottom for vertically-oriented boxes, and right for horizontally-oriented boxes.) The final box is just for padding.

I then close off the `<bevel>`, `<template>` and `<xwt>` tags. The widget is done. I can now put a `<statusline id="mystatus"/>` tag in an XWT file and write status messages by setting the text property. `$mystatus.text="Login Failed.";` `$mystatus.icon = "org.xwt.icons.stop";` and so on.

<Heading name="Traps">

Traps are just functions placed on box properties. There are two types of trap – a read trap and a write trap. Write traps are called when something writes to the box' s property and are designated by a single underscore. `_text`, `_color`, and so on are examples of write traps. If the property is read, the trap is not called, the value of the property is simply returned.

```
<box id="mybox">
    _foo = function(z) {
        xwt.println("foo is " + z);
    }
</box>
```

Now if I say `$mybox.foo = "blah";` you will see "foo is blah" written to the console. If I put this code snippet in a bigger XWT file and set the same kind of trap on the outer widget' s `foo` property, I will get two lines printed – one for the trap that executed on the outer box' s property, and then the trap which was placed on the `foo` property of my box shown above. The traps will execute (or cascade) in reverse order – the most recently added trap to the oldest.

Read traps are called whenever the property is read *or* written. As shown in the sample code, you differentiate between a read and a write to the property by looking at whether the trap was called with an argument since reading from an property doesn' t provide any arguments. Read traps are designated with a double-underscore. `__ReadTrap` is an example read trap. Read traps do not automatically cascade like write traps do, so they are often used to prevent changes from being seen in lower levels of the widget or to allow a widget to manually cascade the trap to lower levels with a different value and effectively "lie" to the lower levels of the widget. This is useful if you need to do some kind of slight-of-hand to keep things sane (keypresses, mouseclicks, etc.).

<Heading name="Talking to the Outside World"/>

XWT can communicate with any XML-RPC or SOAP server. I personally use `Frontier::RPC2` with Perl. On the Perl side I write a simple XML-RPC server and export a function or two:

```
#!/usr/bin/perl -w

use Frontier::Daemon;

sub FuncOne
{
 print "FuncOne() called\n";

 my $text = `cat /home/andrew/blah.txt`;
 return $text;
}

# Call me as http://localhost:8080/RPC2

 my $methods = {
                'funcOne' => \&FuncOne,
               };

 Frontier::Daemon->new(LocalPort => 8080, methods => $methods)
 or die "Couldn't start HTTP server: $!";
```

Simple enough. Now in the XWT world, I would create a trap on a button click and send off an XML-RPC request in a background thread:

```
$connectButton._Click1 = function() {
    xwt.thread = function() {
        var ret = xwt.xmlrpc("http://localhost:8080/RPC2").funcOne();
    }
}
```

All communications occur in background threads. You can wrap `try` and `catch` blocks around it to try and catch errors as well:

```
$connectButton._Click1 = function() {
    xwt.thread = function() {
        try {
            var server = xwt.xmlrpc("http://localhost:8080/RPC2");
            ret = server.getHTML();
        } catch (e) {
            xwt.recursivePrintObject(e);
        }

        var dom = xwt.parseHTML(ret);
        xwt.recursivePrintObject(dom);
        $html.dom = dom;
        }
    }
}
```

The latter example is using the HTML widget, with which you can display HTML data easily. You simply call the supplied `parseHTML()` function and then point the HTML widget to the resultant DOM.

<Heading name="In this corner..."/>

XWT isn' t the only cross-platform windowing toolkit. I haven' t done extensive work with the others, but I will try to present a one-sided view of why I rejected them and chose XWT. :-)

Mozilla's XUL – I didn' t want a huge environment and to be locked to one browser. XUL ties me to the Mozilla codebase. It does, however, also offer pretty much everything XWT offers if you already use Mozilla.

W3C DOM – While DOM is cross-platform and tries to be standard, it is slow and cumbersome whenever it comes to doing anything moderately complex and is just plain difficult or even impossible to do things like columnar lists, borderless windows, getting consistent font behaviour across platforms or designing things like tree lists, tear-off tabs and other nontrivial widgets. DOM utilizes a complex tree structure and HTML doesn' t have very good support for dynamic properties to begin with. JavaScript and DHTML seem to fight each other on a lot of things, and the marriage of the two just isn' t blissful.

Macromedia Flash – The authoring tools need to be purchased. Creating reusable modular widgets is not easy to do in Flash. The widgets are not themeable, and the engine is only partially available and peripherally maintained under Linux. All in all, it' s a proprietary system for creating dynamic graphics and presentations, not a robust structure for application development.

Java/Swing – You need a "real" programmer to implement anything remotely useful in Java. The development time for creating things in Swing is much longer than it would be in XWT. XWT already has native clients for Linux and Win32, with MacOS X coming, using the actual Java bytecode engine as a fallback since it is slower.

CURL – same type of idea as XWT but it has per-user licensing.

<Heading name="XWAR Files"/>

XWT applications are downloaded in XWt ARchives (XWAR) files. These are simple infozip .zip files with all of the individual .xwt files that make up an application contained within. One of my example .xwar files looks like this:

```
$ unzip -t html.xwar
Archive:  html.xwar
    testing: org/                     OK
    testing: org/xwt/                 OK
    testing: org/xwt/html/            OK
    testing: org/xwt/html/p.xwt       OK
    testing: org/xwt/html/test.xwt    OK
    testing: main.xwt                 OK
```

As you can see, nothing unusual, just a hierarchical list of all the files (widgets) required to bring up the application. This file can be put on any web server. Properly written, the critical code and data is kept on a server and accessed via XML-RPC so giving away the source to your GUI to anyone who runs it doesn' t compromise the security of your data or your business logic.

The recommended method for running .xwar files is through xwt.org' s launcher:

```
   http://launch.xwt.org/stable/www.mysite.com/myxwar.xwar
```

`launch.xwt.org` is the website provided to launch any .xwar from anywhere using XWT' s signed launcher applet. IE and Mozilla both work well with this, but Konqueror doesn' t yet support signed applets so the XWT launcher provides a "shoehorn" Java applet for most browsers – an applet which gets the XWT engine on the computer and then launches it, telling it to go to www.mysite.com and download and run myxwar.xwar.

If you' re actively developing an XWT application, chances are that you don' t want to bother with all of this. You can download the latest (or any previous) version of the XWT engine at www.xwt.org/dist and then run it locally:

```
./xwtlinux /path/to/my/dev/dir/
```

where `/path/to/my/dev/dir/` is the directory that `main.xwt` resides in.

<Heading name="Future"/>

There is some preliminary work already done which allows XWT to utilize Scalable Vector Graphics (SVG) – essentially removing the boundaries of a 96-dpi environment. SVG boxes can be sized, rotated and projected without loss of resolution. Fonts will be heading this way as well – currently custom fonts for XWT are made up of a plain graphics format as outlined in the reference section of xwt.org. XWT apps will look great on the monitor or on your 1200DPI laser printer.

Two-way communications between the server and XWT app is also in the works. Currently communications are initiated one-way only (XWT app -> server).

The XWT team is also working on a local launcher app for those of us who don' t like the idea of xwt.org knowing every xwar that is launched. The launcher app will sit in your systray and XWT applications will contact it for user preferences, the XWT engine itself, and other goodies.

<Heading name="Summary and Links"/>

I was introduced to XWT (XML Windowing Toolkit) about ten months ago – I saw it mentioned on either Slashdot.org or Kuro5hin.org in a story about cross-platform windowing toolkits. I' ve been hooked ever since. It seems to have all the power I need and the ability to interface to any XML-RPC or SOAP server is very exciting. I' m not tied to a single vendor, and I' m not tied into some proprietary system.

http://www.xwt.org
http://wiki.xwt.org
http://lists.xwt.org
irc.freenode.net #xwt

`[end]`