Interactive Web Physics, Version 3.
Tutorial: Creating a new DObject type
2007-Feb-15 Brockman

This document is designed to walk you through the steps necessary to create a new object type in IWP 3. One of the main features of version 3 is that the object system was completely revamped to make it easy to plug in new object types. There is a series of interfaces that live in edu.ncssm.iwp.plugin.* that your object must implement to be designed, xml saved, xml loaded, animated, and calculated. There is also an object Factory where you must list your object so it appears in the 'new' menus.

Pick the object Name: FloatingText

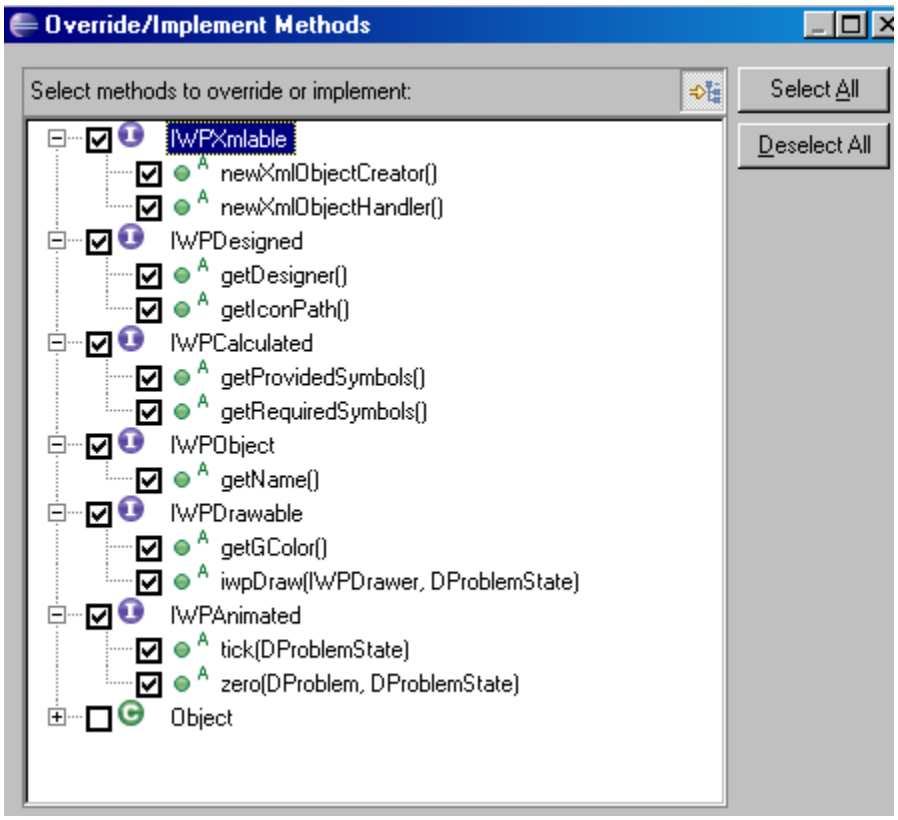
Create a new package: edu.ncssm.iwp.objects.floatingtext

Create a new class: edu.ncssm.iwp.objects.floatingtext.DObject_FloatingText
Superclass = Object

Set Implements to:

```
implements IWPObject, IWPAnimated, IWPDrawable, IWPCalculated, IWPDesigned, IWPXmlable
```

In Eclipse: Source -> Override Implement Methods:



You'll see all the methods you have to implement for the object to be complete.

Pick the attributes that your object is going to have. You shoud default them to values or elese you're going to get null poitner exceptions. The sets here are going tobe the default values of a new object in the designer.
Floating Text is going to have:

```
String name = "New_FloatingText";
String text = "";
MCalculator value = new MCalculator_Parametric("0");
GColor fontColor = new GColor();
int fontSize = 1;
MCalculator xpath = new MCalculator_Parametric("0");
MCalculator ypath = new Mcalculator_Parametric("0");

public String getName() { return name; }
public GColor getGColor() { return fontColor; }
```

With these primitives, I can satisfy part of the Dobject_Designer interface

I also did a source, generate getters and setters so I can use accessors instead of variables to get + set values. For example myNewFloatingText.setText("hello world"). Preferred java way.

Xmlable

Now, its' time to do the XML. I would recommend copying the Dobject_FloatingText_XMLCreator and XMLHandler as Starting points. These class uses the Sub xml handleres for Mcalcualtors and Gcolors, as well as handling serialization and deserialization of Primitives like int and String. The Parser we use is a SAX2 style parser called piccollo, designed to be small + applet packagable.

Designable

```
public String getIconPath()
{
	return "images/icon_DObject_FloatingText.gif";
}
```

And I also recommend copying Dobject_FloatingText_designer. This sets up the swing designer to be able to edit your object properties visually. There are sub designers that you should use to to do things like color + calculators.

You must define the getIconPath, or else you get a null pointer like this:

```
at java.util.Hashtable.get(Hashtable.java:334)
	at edu.ncssm.iwp.ui.GIconSet.cacheLoad(GIconSet.java:92)
	at edu.ncssm.iwp.ui.GIconSet.getObjectIcon(GIconSet.java:38)
	at
edu.ncssm.iwp.plugin.test.TEST_IWPObject.constructDesignedPanel(TEST_IWPObject.java
:196)
```

Calculated:
The calculated interface helps the animator resolve variable and symbol dependencies between objects. Your object must list out all the variables that it requires and all the symbols that it provides. This is usually best found form the calculator properties of the objects.

A very simple case.

```
	// Interface: Calculated
	public Collection getProvidedSymbols() throws InvalidEquationException
	{
		ArrayList out = new ArrayList();
		out.add(getName());
		out.addAll(xpath.getProvidedSymbols());
		out.addAll(ypath.getProvidedSymbols());
		out.addAll(value.getProvidedSymbols());
		return out;
	}

	public Collection getRequiredSymbols() throws InvalidEquationException
	{
		ArrayList out = new ArrayList();
		out.addAll(xpath.getRequiredSymbols());
		out.addAll(ypath.getRequiredSymbols());
		out.addAll(value.getRequiredSymbols());
		return out;
	}
```

For a more complex case, check out Dobject_Time – he is responsible for setting the variable 't' into the equation, and is always executed first by the Animator.

Animated:
This supports the tick and reset methods. The animator calls these when the user hits reset and when the ticker thread advances the time state to the next Tick. In both cases, the DproblemState is passed, which contains the MvariabelsHistory. This variable history contains the data for all previous frames of animation and is required to calculate your new objects current frame. Objects always calculate looking at the current frames variables – the animator is responsible for ordering them at the core level.

These methods must set into the variables hash the symbols that are exposed to other parts of the problem.

```
// Animated
	public void tick(DProblemState state)
		throws UnknownVariableException, UnknownTickException,
InvalidEquationException
	{

		state.vars().setAtCurrentTick(getName() + ".x"+MCalculator.SYMBOL_DISP,
xpath.calculate(state.vars()) );
		state.vars().setAtCurrentTick(getName() + ".y"+MCalculator.SYMBOL_DISP,
ypath.calculate(state.vars()) );

		double calcValue = value.calculate(state.vars());
		state.vars().setAtCurrentTick(getName(), calcValue );
```

```
            state.vars().setAtCurrentTick(getName() + ".value", calcValue );


      }

      public void zero(DProblem problem, DProblemState state)
            throws UnknownVariableException, InvalidEquationException,
UnknownTickException
      {
            tick(state);
      }
```

Drawable:

Draws the current state onto the canvas. This method shoulnd't do any calculation, instead it should just pull the values out of the supplied DprolbemState.vars

```
      // Drawable
      public void iwpDraw(IWPDrawer drawer, DProblemState state)
            throws UnknownVariableException, UnknownTickException,
InvalidEquationException
      {
            double nowValue = state.vars().getAtCurrentTick(getName());
            String text = this.getText() + " = " + nowValue;

            drawer.drawString ( text,

      state.vars().getAtCurrentTick(getName()+".x"+MCalculator.SYMBOL_DISP),

      state.vars().getAtCurrentTick(getName()+".y"+MCalculator.SYMBOL_DISP) );
      }
```

Plugin Factory:

open up ed.ncssm.iwp.plugin.IWPPluginFactory, and add your Object Name to the pluggedObjectNamesOrdered and pluggedObjects Hashtable. Example:

```
 private static String[] pluggedObjectNamesOrdered = { "Solid", "Input", "Output",
"WaveBox", "FloatingText" };

 pluggedObjects.put ( "FloatingText",
"edu.ncssm.iwp.objects.floatingtext.DObject_FloatingText" );
```

IWP3 uses dynamic class loading to pull these out of the current classloader space (usually the packaged .jar).

With this added here, you should be able to run the Test harness for your object. Run as Java program: edu.ncssm.iwp.plguin.test.TEST_IWPObject. No Arguments. You should see:

Clicking the Floating Text will spawn a 'new' version of your object and allow you to see all the interfaces that it supports, and interact with the widget parts of IWP that manipulate those interfaces. At time of writing, I focused on the Xmlable and Designable interfaces.

Testing with TEST_IWPObject

Designable:
Add some paths, and a value equation and some text. Make sure Icon shows at top left.

Xmlable:
The left panel + right panel are the same, just for conveneince there are two.
Click left panel 'create' – this will output the xml for your object.
Click handle, this will re-create the object based on the xml parsing. At a basic level, your object
Click create again and the next should NOT change.

Animated:
Create a new guy, define some paths, and see home move around the screen. Slide the time to see the paths take effect.

Creating a Packacked TEST_NewObject Problem.

When you create a new object, please take the time to create a few packaged test problems and add to the /etc/packagedProblems/directory.xml TEST category.

See etc/packagedProblems/TEST_FloatingText.xml