

CatalogAPI

So you want something to manage your datastores (rather than use singletons), and do cross datastore operations.

Summary:

1. Backport the uDig Catalog API ...

Future Work:

1. Drop in extensible metadata
2. Use Filter 1.1 to bring it in line with the current OGC standards

Of course only the first one is needed to get things going.

Status

The backport is currently finished. There is a [Catalog Tutorial](#) for an in depth description of the API and a tutorial.

Background

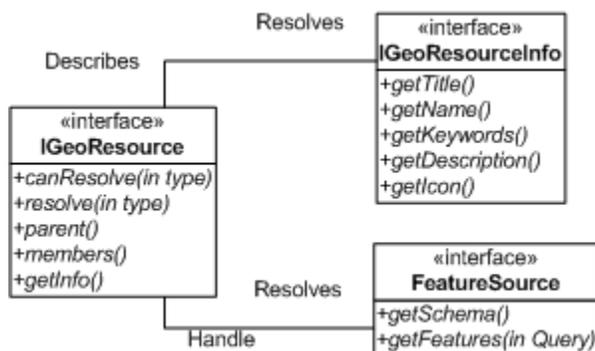
UDIG Catalog has a long history:

- started as [GeoServer Data API](#)
- and was turned into the [GeoTools Repository Interface](#) to allow GeoTools modules ([like validation](#)) to work directly against the GeoServer catalog (or any other application)
- attempt was made to port this to the Catalog standard for GeoAPI, a [Catalog Interface](#) was created and used to supplement GridCoverageExchange, the standards were not ready yet and org.opengis.catalog was withdrawn

Catalog API

A working Java 5 API that is a joy to use (written by David Zwiers):

- <http://udig.refractions.net/docs/api-udig/net/refractions/udig/catalog/package-summary.html>



This currently supports DataStores, WFS and WMS and GridCoverages in a unified manner. The API can easily be extended to reveal and query internal structure (WMS can display nested layers, a grid coverage can support multiple resolutions).

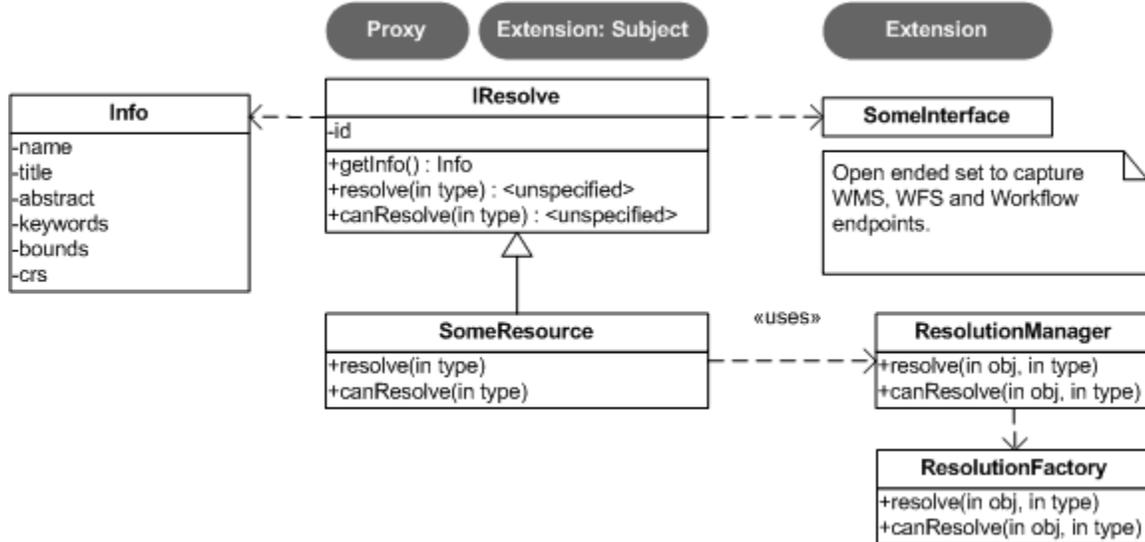
Docs:

[-Catalog \(Developers Guide\)](#)

Fixes

Resolution Manager

We need to include an extension mechanism - see the following picture:



Basically we cannot assume as library designers what manager of class/interface client code wants to interact with the resource through. As an example a raster may be accessed through GeoTools GridCoverage or via JOSSIM. A less extreme example is Shapefile and IndexedShapefile.

Nested WMS Layers

The WMS Layer is a GeoResource, that is allowed other layers as children, while already accounted for in the API - the implementation on udig trunk does not make use of this.

Having a working example of this idea with WMSLayer will "head off" people making bad assumptions.

Related Idea: Databases group their contents into schemas - these almost act as a Folder for GeoResources, they are used to group data but are not data in themselves.

There has been one uDig request for "abstract" WMS Layers, I think they also represent the "folder" idea.

Bonus

On Metadata

An idea on how to do metadata:

- http://weblogs.java.net/blog/jive/archive/2005/07/mad_metadata_pl.html

In short, udig [IGeoResourceInfo](#) and geosever [FeatureTypeInfo](#) are both based on a minimal take on dublin core. Above link explains how we can transition between Metadata standards as required, and more importantly flow an open ended set of metadata through our object based system. The technique supports XML or Object based systems, and is way cool.

On Filter

An idea of how to move "Filter" forward ...

-<http://weblogs.java.net/blog/jive/archive/2005/07/xml_standards_a.html>

Filter is used as the query language for Features and Metadata (The Catalog 2 spec is expressed in terms of Filter 1.1). Right now the udig catalog support basic keyword, location lookup. Our filter implementation is only against Features. Both things are totally fixable.

Status:

- <http://svn.geotools.org/geotools/trunk/spike/gtapi/>

Issues

IProgressMonitor vs. ProgressListener

The uDig Catalog API uses Eclipse's IProgressMonitor to signal progress of lengthy operations to interested classes or to the user. This very same task is done by org.geotools.util.ProgressListener in GeoTools. However the two interfaces have fundamental differences:

ProgressListener:

```
public abstract String getDescription();
public abstract void setDescription(final
String description);
public abstract void started();
public abstract void progress(final float
percent);
public abstract void complete();
public abstract void dispose();
public abstract void
warningOccurred(String source, String
margin, String warning);
public abstract void
exceptionOccurred(final Throwable
exception);
```

IProgressMonitor:

```
public void beginTask(String name, int
totalWork); (totalWork can be -1 = unknown)
public void done();
public boolean isCanceled();
public void setCanceled(boolean value);
public void setTaskName(String name);
public void subTask(String name);
public void worked(int work);
(public void internalWorked(double work);)
```

One difference is that ProgressListener (as well as the Swing ProgressMonitor BTW) work cumulative, whereas a caller to IProgressMonitor just tells how much work was just done, not how much total work is already done. Now, when uDig code gets backported the implementations must work against ProgressListener instead of against

IProgressMonitor. That is, they themselves have to keep track of the total amount of work done. On the other side The adapter from ProgressListener to IProgressMonitor has to compute the progress difference since the last call and send this to IProgressMonitor.

Other differences are:

1. ProgressListener can handle warnings and exceptions. Since IProgressMonitor doesn't, the wrapper will have to care about how to signal these warnings to the user.
2. IProgressMonitor supports the notion of subtasks, but this can be simulated.
3. ProgressListener supports no possibility to explicitly cancel the task.

What interfaces/classes to backport to GeoTools

Following interfaces seem mandatory:

- IGeoResource (becomes GeoResource?)
- IGeoResourceInfo (becomes GeoResourceInfo?)
- IResolve (becomes Resolve?)

Further suggestions to backport:

- IService
Jody: I think we do need a way to manage services, it may not seem required for rendering - but you will run into the situation where two layers come from the same WMS and can be handled with a single request (when they are next to each other in zorder). Also you need the IService to "hold" the WMS without duplication.
- ICatalog???
Jody: You could consider a catalog api without searching, it is mainly used to manage conentions and provides an event system. That is reporting on status changes as services are connected, disconnected, fixed etc ...

Info Objects and Images

What format should images returned by info objects. The following is the uDig implementation:

```
public class IServiceInfo {
    ...
    public
    org.eclipse.jface.resource.ImageDescriptor
    getIcon();
}
```

Possible Solutions:

Return a swing image and have udig adapt it to SWT.

Pain for the uDig developers. Doesn't really fit in with the idea of Lazy Loading.

Return a swing Icon interface and adapt it to SWT.

Sounds similar, but Icon does not say anything about whether the implementation has the image ready or uses lazy loading. Both is possible.

I have an adapter class for this which supports lazy loading. It is attached: [SwingImageDescriptor.java](#)

The idea is: API returns Icon interface, Icon gets wrapped in SWT image descriptor. Only when SWT image is to be created will the adapter ask the icon to paint itself. Then the icon implementation fetches the data and paints. So lazy loading should be fine.

That's what I suggest Matthias Basler

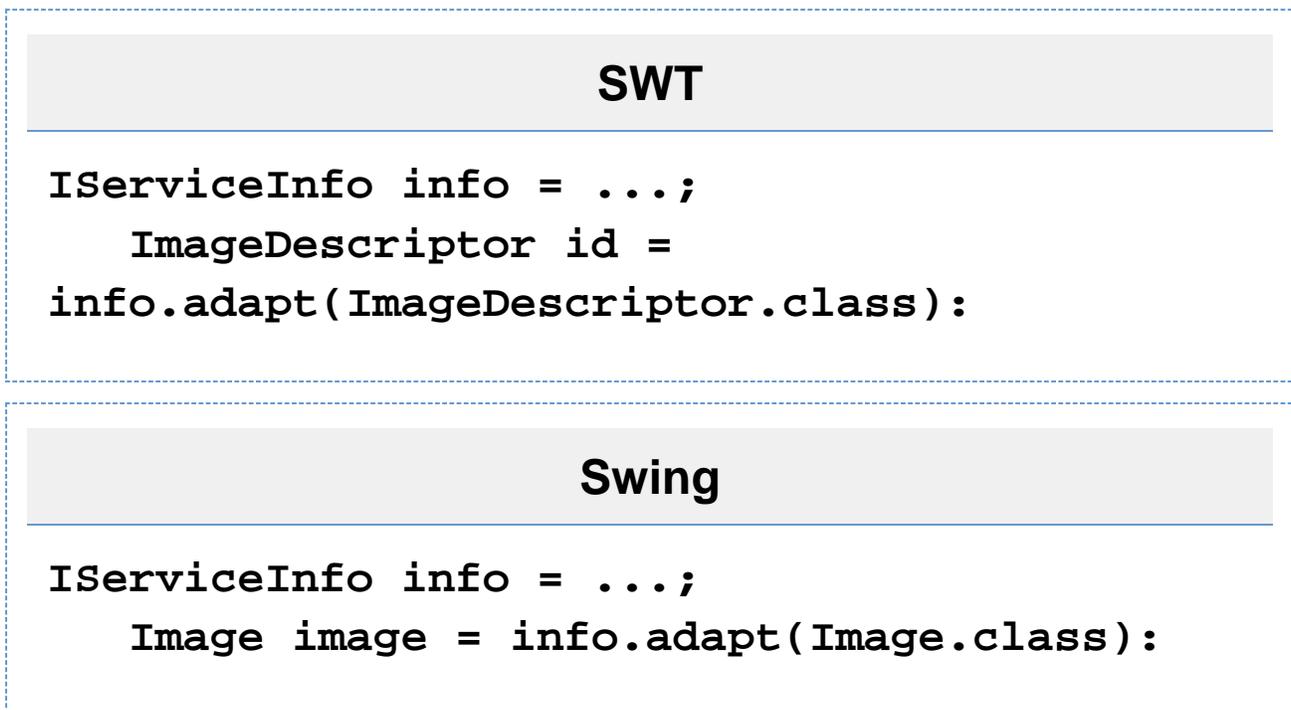
Remove the capability to get an icon from the interface.

Pain for client code. Images must be obtained through a separate api or just by hacking.

Not an option. This case occurs several times in GeoTools/GeoWidgets (f.e. in map and map layer) and it is impossible to remove any icons from GeoTools just because Eclipse can't use them without adapter.

Have info objects implement an IAdaptable pattern that does block.

Client code can do things like:



The pain here is that different IServiceInfo objects are needed for different platforms (SWT vs Swing).

If it is only that ... that's what inheritance is good for.

I fear however that it tends towards a non-intuitive API. Matthias

Consequently avoid both Swing and SWT and use own interfaces + adapters

Makes only sense if done consequently, because then it does not force the Swing user to have the SWT library, nor

does it force the SWT user to have Sun Java installed.

Pain is, that this means reinventing the wheel (We'd have to write our own Icon and ImageDescriptor classes and others.) Moreover the API would become strange to the normal user.

? Question: How does GeoWidgets handle this one?

i Original intention for GeoWidgets was to use AWT only (i.e. nothing Sun specific) and adapt to SWT, but probably there's no point to circumvent Swing: A) Most people have it anyway and B) It is widely used in GeoTools, isn't it? So I used the javax.swing.Icon interface and adapted SWT.

i Works for me, that is what the catalog implementation will use. I believe uDig already has an implementation of ImageDescriptor which implements javax.swing.Icon.

Further comments appreciated.