MS RFC 3: Feature Layer Plug-in Architecture

Date:	2005/08/22
Author:	Jani Averbach
Contact:	javerbach at extendthereach.com
Last Edited:	2008-12-23
Status:	Adopted
Version:	MapServer 4.8
Id:	ms-rfc-3.txt 8278 2008-12-23 21:34:31Z hobu

Abstract Solution

Implement a virtual table structure for `layerObj`. This structure will contain function pointers for all layer specific operations. This structure will be populated when the layer is opened or first time accessed. After that all back-end format specific layer operations will happen through this vtable which is cached in the `layerObj` struct.

Technical Solution

All file names and numbers are against released MapServer 4.6.0 source code.

1. Add new field to the `layerObj`. It will be pointer to the vtable, which will contain function pointers for this layer.
2. Add the virtual table architecture

   The vtable will be initialized when the layer is accessed at the first time. The vtable will be populated with dummies when it is created and these dummies will implement feasible default actions if there is any (e.g. do nothing) or return error by default.

   2.1. Standard functions which are found currently from `maplayer.c`

   2.1.1. InitItemInfo

   ```
   int (*LayerInitItemInfo)(layerObj *layer);
   ```

   2.1.2. FreeItemInfo

   ```
   void (*LayerFreeItemInfo)(layerObj *layer);
   ```

   2.1.3. Open

   ```
   int (*LayerOpen)(layerObj *layer);
   ```

   Currently there are two layers which accept more than the generic layer arg for LayerOpen function:

   ```
   int msOGRLayerOpen(layerObj *layer,
                      const char *pszOverrideConnection);
   int msWFSLayerOpen(layerObj *lp,
                      const char *pszGMLFilename,
                      rectObj *defaultBBOX);
   ```

However, these are called from `msLayerOpen` with `NULL` args, so I think that proposed interface for this virtual table function should be fine.

2.1.4. IsOpen

```
int (*LayerIsOpen)(layerObj *layer);
```

2.1.5. WhichShapes

```
int (*LayerWhichShapes)(layerObj *layer,
                        rectObj rect);
```

2.1.6. NextShape

```
int (*LayerNextShape)(layerObj *layer, shapeObj *shape);
```

2.1.7. GetShape

```
int (*LayerGetShape)(layerObj *layer, shapeObj *shape,
                     int tile, long record);
```

2.1.8. LayerClose

```
int (*LayerClose)(layerObj *layer);
```

2.1.9. LayerGetItems

```
int (*LayerGetItems)(layerObj *layer);
```

2.1.10. GetExtent

```
int (*LayerGetExtent)(layerObj *layer,
                      rectObj *extent);
```

2.1.11. GetAutoStyle

```
int (*LayerGetAutoStyle)(mapObj *map, layerObj *layer,
                         classObj *c, int tile,
                         long record);
```

2.2. New functions and/or fields for vtable

2.2.1. CloseConnection

This function is used to actually close the connection, in case that layer implements some kind of connection pooling by its own. If layer doesn't use any connection pooling, this function should be implemented as no-op. Caller should first call layer's "close" function, and finally at the very end `CloseConnection`.

The signature of function is

```
int (*LayerCloseConnection)(layerObj *layer);
```

And the main place where this function will be called, is `mapfile.c: 4822`

```
void msCloseConnections(mapObj *map)
```

This function is needed because e.g. POSTGIS is implementing this usage pattern at the moment maplayer.c:599

```
 void msLayerClose(layerObj *layer)
 ...
/*
 * Due to connection sharing, we need to close the results
 * and free the cursor, but not close the connection.
 */
 msPOSTGISLayerResultClose(layer);
```

2.2.2. SetTimeFilter

This function is used to create a time filter for layer. At the moment we have three special cases (maplayer.c: 1635):

1. POSTGIS with it's own function
2. Layers with backticks delimited expressions
3. Layers without backticks

The idea is provide a generic helper function,

```
int makeTimeFilter(layerObj *lp,
                   const char *timestring,
                   const char *timefield,
                   const bool bBackTicks)
```

And the actual layer's SetTimeFilter could use the above, or implement something totally different as POSTGIS is doing at the moment.

The signature for layer's vtable function is

```
int (*LayerSetTimeFilter)(layerObj *lp,
                          const char *timestring,
                          const char *timefield);
```

2.3. Extra functions to add to the vtable

2.3.1. FLTApplyFilterToLayer (mapogcfilter.c: 1084)

This is the main filter interface for layers. We will provide two helper functions, one for "SQL" case and the another for "non-SQL" case, and set all layers, except POSTGIS, ORACLESPATIAL and OGR to call directly this "non-SQL" version of this helper function (else-branch of if).

ORACLESPATIAL, POSTGIS and OGR could use "SQL" version of the helper function (actual if-branch) if the conditions for this are met, otherwise they will use Non-SQL version of the function.

2.3.2. layerObj->items allocation

There will be vtable function for allocation `items` and initialization for `numitems`. If layer has some special needs for these objects it can override default function.

The signature for function will be:

```
int (*CreateItems)(layerObj *)
```

which does the allocation and set numitems to correct value.

2.3.3. msCheckConnection (`mapfile.c: 4779`)

This API is deprecated. It is called only from `msMYGISLayerOpen`. We will not provide this functionality through vtable.

2.4. Interface functions for internal layers

We have to add some new interface functions to access layers.

2.4.1 Function interface to initialize internal layer type

We need a per layer type a function which will be called when the layer's vtable has to be initialized. These functions will be

```
int msXXXInitializeLayerVirtualTable(layerObj *)
```

where XXX will be name of the layer. This function is called anytime when the vtable isn't initialized and the layer is accessed at the first time.

2.4.2 Function interface to change the connectiontype of layer

To change the connection type of layer, it has to be done by function interface. Accessing directly `connectiontype` field is not supported anymore.

To change the connectiontype and to connect to the new layer, there will be following interface function

```
int msConnectLayer(int connectiontype,
                   const char *library_str)
```

where `connectiontype` is the type of new layer, and `library_str` is the name of library which will prodive functionality for this new layer. For internal layer types this second argument is ignored.

3. Remove unwanted interfaces

Frank Warmerdam proposed [FW1] that we remove all layer specific interface functions from `map.h`.

> I see each "built-in" module such as mapsde.c providing a registration function such as "msSDEInitializeLayerVirtualTable" so that none of the layer type specific definitions need to appear in map.h any more.

Files and objects affected

This proposal will affect at least following files and objects:

- `map.h`
  - `layerObj` will contain new fields. There will be a new object `vtableObj` in the `map.h`.
- `maplayer.c`

    - Various changes, layer specific `switch`-statements will go away, vtable handling and layers vtable initialization will be added.
- `mapfile.c`
    - Cleaning of `msCheckConnection`
    - Vtable for `msCloseConnection`
- `mapogcfilter.c`
    - Remove layer-logic from `FLTApplyFilterToLayer`
- `mapXXX.c`, where `XXX` is the name of layer.
    - Add new initialization function
    - Add all new interface functions
    - Fix existing interface functions, if needed / add wrappers for `msOGRLayerOpen` and `msWFSLayerOpen`.

Backwards compatibility issues

This is binary and source code level backward incompatible change. The proposal will remove some previously public functions, and add new field(s) to the `layerObj` struct.

This proposal is not MapScript backward compatible, it will break scripts which change directly `connectiontype` field in `layerObj`.

The proposal is MAP-file backward compatible.

Implementation Issues

- Biggest problem is probably that the author has ignored or missed something by oversight which will show up during implementation. However, there is a prototype implementation of external plug-in architecture which works at the moment and is based on ideas presented in this proposal. So there is some real life experience that this architecture change is feasible thing to do.
- I also like to note that this proposal won't remove all layer specific code from MapServer e.g. `WMF`, `WMS` and `GRATICULE` are used as special cases from place to place.

Bug ID

Bug 1477

Voting history

Vote proposed by Jani Averbach on 9/19/2005, result was +4 (3 non-voting members).

Voting +1: Howard Butler, Frank Warmerdam, Yewondwossen Assefa, Daniel Morissette

Proposal passes and will move forward.

Open questions

- How do we like to expose layer's virtual table to the layers. We have at least two different routes to go:
    - expose it as a struct, layers will fill vtable pointers by accessing directly struct's field.
    - expose it as a complete opaque type, vtable pointers will be set by accessing functions `setLayerOpenFP`, `setLayerCloseFP` and so on.

The advance of second option is that this way we could easily add new functions to the struct if we refactor code more or found some logic which is ignored by oversight in this proposal.

- Are there any special issues with the raster query layer support which is handled via the layer API?

FW1

```
Frank Warmerdam on the mapserver-dev:
Message-Id: <smtpd.490f.4303f2ee.d8246.1@mtaout-c.tc.umn.edu>
Date:       Wed, 17 Aug 2005 22:31:09 -0400
Subject:    Re: Mapserver Plug-in Infastructure: RFC and PATCH
```