Reference Manual

Generated by Doxygen 1.5.1

Wed Jul 25 09:57:17 2007

Contents

1 Page Index **1**

1.1 Related Pages 1

2 Page Documentation **3**

2.1 Motions of the GDAL/OGR Project Management Committee 3

2.2 RFC 1: Project Management Committee Guidelines 4

2.3 RFC 2: Migration to OSGeo Subversion Repository 6

2.4 RFC 3: GDAL Commiter Guildlines 8

2.5 RFC 4: Geolocation Arrays 12

2.6 RFC 5: Unicode support in GDAL 15

2.7 RFC 6: Geometry and Feature Style as OGR Special Fields 19

2.8 RFC 7: Use VSILFILE for VSI*L Functions 28

2.9 RFC 8: Developer Guidelines 29

2.10 RFC 9: GDAL Paid Maintainer Guidelines 31

2.11 RFC List 33

Chapter 1

Page Index

1.1 Related Pages

Here is a list of all related documentation pages:

Motions of the GDAL/OGR Project Management Committee . . . 3
RFC 1: Project Management Committee Guidelines . . . 4
RFC 2: Migration to OSGeo Subversion Repository . . . 6
RFC 3: GDAL Commiter Guildlines . . . 8
RFC 4: Geolocation Arrays . . . 12
RFC 5: Unicode support in GDAL . . . 15
RFC 6: Geometry and Feature Style as OGR Special Fields . . . 19
RFC 7: Use VSILFILE for VSI*L Functions . . . 28
RFC 8: Developer Guidelines . . . 29
RFC 9: GDAL Paid Maintainer Guidelines . . . 31
RFC List . . . 33

Chapter 2

Page Documentation

2.1 Motions of the GDAL/OGR Project Management Committee

2.1.1 Motions of 2006

- **Mailing List Changes** (approved)

  On Dec 29, 2006, Frank Warmerdam motions to:

  1. To create a `gdal-announce@lists.osgeo.org` mailing list which is moderated and only used for infrequent GDAL/OGR related announcements such as releases, major bugs, and other major events.
  2. To migrate the `gdal-dev@lists.maptools.org` mailing list to lists.osgeo.org in such a way to preserve existing members, and hopefully preserve settings.
  3. To create a `gdal-commits@lists.osgeo.org` mailing list which will be used for SVN commit messages once the SVN transition is complete.
  4. To appoint Frank Warmerdam as lists administrator for the GDAL/OGR mailing lists.

  Motion passed Jan 4, 2007. FrankW(+1), HowardB(+1), DanielM(+1).

- **Project Sponsorship** (approved)

  On Nov 6, 2006, Frank Warmerdam motions for "GDAL/OGR to join the OSGeo Project Sponsorship Program."

  Motion passed Nov 9, 2006. FrankW(+1), HowardB(+1), DanielM(-0).

2.2 RFC 1: Project Management Committee Guidelines

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: Adopted

2.2.1 Summary

This document describes how the GDAL/OGR Project Management Committee determines membership, and makes decisions on GDAL/OGR project issues.

In brief the committee votes on proposals on gdal-dev. Proposals are available for review for at least two days, and a single veto is sufficient to delay progress though ultimately a majority of members can pass a proposal.

2.2.2 Detailed Process

1. Proposals are written up and submitted on the gdal-dev mailing list for discussion and voting, by any interested party, not just committee members.
2. Proposals need to be available for review for at least two business days before a final decision can be made.
3. Respondents may vote "+1" to indicate support for the proposal and a willingness to support implementation.
4. Respondents may vote "-1" to veto a proposal, but must provide clear reasoning and alternate approaches to resolving the problem within the two days.
5. A vote of -0 indicates mild disagreement, but has no effect. A 0 indicates no opinion. A +0 indicate mild support, but has no effect.
6. Anyone may comment on proposals on the list, but only members of the Project Management Committee's votes will be counted.
7. A proposal will be accepted if it receives +2 (including the proposer) and no vetos (-1).
8. If a proposal is vetoed, and it cannot be revised to satisfy all parties, then it can be resubmitted for an override vote in which a majority of all eligible voters indicating +1 is sufficient to pass it. Note that this is a majority of all committee members, not just those who actively vote.
9. Upon completion of discussion and voting the proposer should announce whether they are proceeding (proposal accepted) or are withdrawing their proposal (vetoed).
10. The Chair gets a vote.
11. The Chair is responsible for keeping track of who is a member of the Project Management Committee.
12. Addition and removal of members from the committee, as well as selection of a Chair should be handled as a proposal to the committee. The selection of a new Chair also requires approval of the OSGeo board.
13. The Chair adjudicates in cases of disputes about voting.

2.2.3 When is Vote Required?

- Anything that could cause backward compatibility issues.
- Adding substantial amounts of new code.
- Changing inter-subsystem APIs, or objects.
- Issues of procedure.
- When releases should take place.
- Anything that might be controversial.

2.2.4 Observations

- The Chair is the ultimate adjudicator if things break down.
- The absolute majority rule can be used to override an obstructionist veto, but it is intended that in normal circumstances vetoers need to be convinced to withdraw their veto. We are trying to reach consensus.

2.2.5 Bootstrapping

Frank Warmerdam is declared initial Chair of the Project Management Committee.

Daniel Morissette, Frank Warmerdam, Andrey Kiselev and Howard Butler are declared to be the founding Project Management Committee.

2.3 RFC 2: Migration to OSGeo Subversion Repository

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: Adopted (implementation delayed)

2.3.1 Summary

It is proposed that the GDAL source tree be moved a subversion repository in such a manner as to preserve the history existing in the CVS repository. A 1.3.x branch will be created after automatic updating of the header format.

2.3.2 Details

1. The conversion will be done by Howard Butler using the cvs2svn tool.
2. At least 24 hours notice will be provided before the transition starts to allow committers to commit any outstanding work that is ready to into the repository.
3. When the conversion starts, the GDAL (and gdalautotest) trees will be removed from cvs.maptools.org, and archived to avoid any confusion.
4. Frank Warmerdam will modify the "daily cvs snapshot" capability to work from SVN.
5. Frank will be responsible for updating the source control information in the documentation.
6. All source files in SVN will have the svn:keywords property set to "Id" by Frank after they are created.
7. Committers will need to get a login on osgeo.org and notify Frank to regain commit access. Committer access on the new repository will be enabled after the above changes are all complete.
8. The GDAL committers document should be updated, removing non-GDAL commiters (ie. libtiff, geotiff, etc).

2.3.3 Header Format

SVN does not support history insertion in source files, and to keep the old history listings around without keeping them up to date would be very confusing. So it is proposed that Frank Warmerdam write a script to strip the history logs out. Changing this:

```
/******************************************************************************
 * $Id: RFC2_SVN.dox 9931 2006-07-26 16:26:39Z fwarmerdam $
 *
 * Project:  GDAL Core
 * Purpose:  Color table implementation.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 2000, Frank Warmerdam
...
 ******************************************************************************
 * $Lcg: RFC2_SVN.dox,v $
 * Revision 1.6  2006/03/28 14:49:56  fwarmerdam
```

```
 * updated contact info
 *
 * Revision 1.5  2005/09/05 19:29:29  fwarmerdam
 * minor formatting fix
 */

#include "gdal_priv.h"

CPL_CVSID("$Id: RFC2_SVN.dox 9931 2006-07-26 16:26:39Z fwarmerdam $");
```

to this:

```
/******************************************************************************
 * $Id: RFC2_SVN.dox 9931 2006-07-26 16:26:39Z fwarmerdam $
 *
 * Project:  GDAL Core
 * Purpose:  Color table implementation.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 2000, Frank Warmerdam
...
 ******************************************************************************/

#include "gdal_priv.h"

CPL_CVSID("$Id: RFC2_SVN.dox 9931 2006-07-26 16:26:39Z fwarmerdam $");
```

2.3.4 Branch for 1.3

Once the headers have been updated appropriately, a 1.3 branch will be established in subversion. The intent is that further 1.3.x releases would be made against this "stable branch" while trunk work is towards a 1.4.0 release targetted for around the time of the OSGeo conference.

2.4 RFC 3: GDAL Commiter Guildlines

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: Adopted

2.4.1 Purpose

To formalize SVN (or CVS) commit access, and specify some guidelines for SVN committers.

2.4.2 Election to SVN Commit Access

Permission for SVN commit access shall be provided to new developers only if accepted by the GDAL/OGR Project Steering Committee. A proposal should be written to the PSC for new committers and voted on normally. It is not necessary to write an RFC document for these votes, a proposal to gdal-dev is sufficient.

Removal of SVN commit access should be handled by the same process.

The new committer should have demonstrated commitment to GDAL/OGR and knowledge of the GDAL/OGR source code and processes to the committee's satisfaction, usually by reporting bugs, submitting patches, and/or actively participating in the GDAL/OGR mailing list(s).

The new committer should also be prepared to support any new feature or changes that he/she commits to the GDAL/OGR source tree in future releases, or to find someone to which to delegate responsibility for them if he/she stops being available to support the portions of code that he/she is responsible for.

All committers should also be a member of the gdal-dev mailing list so they can stay informed on policies, technical developments and release preparation.

New commiters are responsible for having read, and understood this document.

2.4.3 Committer Tracking

A list of all project committers will be kept in the main gdal directory (called COMMITERS) listing for each SVN committer:

- Userid: the id that will appear in the SVN logs for this person.
- Full name: the users actual name.
- Email address: A current email address at which the committer can be reached. It may be altered in normal ways to make it harder to auto-harvest.
- A brief indication of areas of responsibility.

2.4.4 SVN Administrator

One member of the Project Steering Committee will be designed the SVN Administrator. That person will be responsible for giving SVN commit access to folks, updating the COMMITERS file, and other SVN related management. That person will need login access on the SVN server of course.

Initially Frank Warmerdam will be the SVN Adminstrator.

2.4.5 SVN Commit Practices

The following are considered good SVN commit practices for the GDAL/OGR project.

- Use meaningful descriptions for SVN commit log entries.
- Add a bug reference like "(bug 1232)" at the end of SVN commit log entries when committing changes related to a bug in bugzilla.
- Changes should not be committed in stable branches without a corresponding bug id. Any change worth pushing into the stable version is worth a bug entry.
- Never commit new features to a stable branch without permission of the PSC or release manager. Normally only fixes should go into stable branches. New features go in the main development trunk.
- Only bug fixes should be committed to the code during pre-release code freeze, without permission from the PSC or release manager.
- Significant changes to the main development version should be discussed on the gdal-dev list before you make them, and larger changes will require a RFC approved by the PSC.
- Do not create new branches without the approval of the PSC. Release managers are assumed to have permission to create a branch.
- All source code in SVN should be in Unix text format as opposed to DOS text mode.
- When committing new features or significant changes to existing source code, the committer should take reasonable measures to insure that the source code continues to build and work on the most commonly supported platforms (currently Linux and Windows), either by testing on those platforms directly, running buildbot tests, or by getting help from other developers working on those platforms. If new files or library dependencies are added, then the configure.in, Makefile.in, Makefile.vc and related documentations should be kept up to date.

2.4.6 Legal

Commiters are the front line gatekeepers to keep the code base clear of improperly contributed code. It is important to the GDAL/OGR users, developers and the OSGeo foundation to avoid contributing any code to the project without it being clearly licensed under the project license.

Generally speaking the key issues are that those providing code to be included in the repository understand that the code will be released under the MIT/X license, and that the person providing the code has the right to contribute the code. For the commiter themselves understanding about the license is hopefully clear. For other contributors, the commiter should verify the understanding unless the commiter is very comfortable that the contributor understands the license (for instance frequent contributors).

If the contribution was developed on behalf of an employer (on work time, as part of a work project, etc) then it is important that an appropriate representative of the employer understand that the code will be contributed under the MIT/X license. The arrangement should be cleared with an authorized supervisor/manager, etc.

The code should be developed by the contributor, or the code should be from a source which can be rightfully contributed such as from the public domain, or from an open source project under a compatible license.

All unusual situations need to be discussed and/or documented.

Commiters should adhere to the following guidelines, and may be personally legally liable for improperly contributing code to the source repository:

- Make sure the contributor (and possibly employer) is aware of the contribution terms.
- Code coming from a source other than the contributor (such as adapted from another project) should be clearly marked as to the original source, copyright holders, license terms and so forth. This information can be in the file headers, but should also be added to the project licensing file if not exactly matching normal project licensing (gdal/LICENSE.txt).
- Existing copyright headers and license text should never be stripped from a file. If a copyright holder wishes to give up copyright they must do so in writing to the foundation before copyright messages are removed. If license terms are changed it has to be by agreement (written in email is ok) of the copyright holders.
- When substantial contributions are added to a file (such as substantial patches) the author/contributor should be added to the list of copyright holders for the file.
- If there is uncertainty about whether a change it proper to contribute to the code base, please seek more information from the project steering committee, or the foundation legal counsel.

2.4.7 Bootstraping

The following existing commiters will be considered authorized GDAL/OGR committers as long as they each review the commiter guidelines, and agree to adhere to them. The SVN administrator will be responsible for checking with each person.

- Daniel Morissette
- Frank Warmerdam
- Gillian Walter
- Andrey Kiselev
- Alessandro Amici
- Kor de Jong
- Howard Butler
- Lichun Wang
- Norman Vine
- Ken Melero
- Kevin Ruland
- Marek Brudka
- Pirmin Kalberer

- Steve Soule
- Frans van der Bergh
- Denis Nadeau
- Oleg Semykin
- Julien-Samuel Lacroix
- Daniel Wallner
- Charles F. I. Savage
- Mateusz Loskot
- Peter Nagy
- Simon Perkins
- Radim Blazek
- Steve Halasz
- Nacho Brodin
- Benjamin Collins
- Ivan Lucena
- Ari Jolma
- Tamas Szekeres

2.5 RFC 4: Geolocation Arrays

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: Development

2.5.1 Summary

It is proposed that GDAL support an additional mechanism for geolocation of imagery based on large arrays of points associating pixels and lines with geolocation coordinates. These arrays would be represented as raster bands themselves.

It is common in AVHRR, Envisat, HDF and netCDF data products to distribute geolocation for raw or projected data in this manner, and current approaches to representing this as very large numbers of GCPs, or greatly subsampling the geolocation information to provide more reasonable numbres of GCPs are inadequate for many applications.

2.5.2 Geolocation Domain Metadata

Datasets with geolocation information will include the following dataset level metadata items in the "GEOLOCATION" domain to identify the geolocation arrays, and the details of the coordinate system and relationship back to the original pixels and lines.

- SRS: wkt encoding of spatial reference system.
- X_DATASET: dataset name (defaults to same dataset if not specified)
- X_BAND: band number within X_DATASET.
- Y_DATASET: dataset name (defaults to same dataset if not specified)
- Y_BAND: band number within Y_DATASET.
- Z_DATASET: dataset name (defaults to same dataset if not specified)
- Z_BAND: band number within Z_DATASET. (optional)
- PIXEL_OFFSET: pixel offset into geo-located data of left geolocation pixel
- LINE_OFFSET: line offset into geo-located data of top geolocation pixel
- PIXEL_STEP: each geolocation pixel represents this many geolocated pixels.
- LINE_STEP: each geolocation pixel represents this many geolocated lines.

In the common case where two of the bands of a dataset are actually latitude and longitude, and so the geolocation arrays are the same size as the base image, the metadata might look like:

```
SRS: GEOGCS...
X_BAND: 2
Y_BAND: 3
PIXEL_OFFSET: 0
LINE_OFFSET: 0
PIXEL_STEP: 1
LINE_STEP: 1
```

For AVHRR datasets, there are only 11 points, but for every line. So the result for a LAC dataset might look like:

```
SRS: GEOGCS...
X_DATASET: L1BGCPS:n12gac10bit.l1b
X_BAND: 1
Y_DATASET: L1BGCPS:n12gac10bit.l1b
Y_BAND: 2
PIXEL_OFFSET: 25
LINE_OFFSET: 0
PIXEL_STEP: 40
LINE_STEP: 1
```

This assumes the L1B driver is modified to support the special access to GCPs as bands using the L1BGCPS: prefix.

2.5.3 Updating Drivers

1. HDF4: Client needs mandate immediate incorporation of geolocation array support in the HDF4 driver (specifially for swath products). (complete)
2. HDF5: Some HDF5 products include geolocation information that should be handled as arrays. No timetable for update.
3. AVHRR: Has 11 known location per-scanline. These are currently substantially downsampled and returned as GCPs, but this format would be an excellent canidate for treating as geolocation arrays. Planned in near future.
4. Envisat: Envisat raw products use geolocation information currently subsampled as GCPs, good candidate for upgrade. No timetable for update.
5. netCDF: NetCDF files can have differently varying maps in x and y directions which could be represented as geolocation arrays when they are irregular. No timetable for update.
6. OPeNDAP: Can have differently varying maps in x and y directions which could be represented as geolocation arrays when they are irregular. No timetable for update.

2.5.4 Changes to Warp API and gdalwarp

Introduce a new geolocation array based transformation method, following the existing GDALTransformer mechanism. A geolocation array transformer will be created with the following function call. The "char **" array is the list of metadata from the GEOLOCATION metadata domain.

void *GDALCreateGeoLocTransformer(GDALDatasetH hBaseDS, char **papszGeolocationInfo, int bReversed);

This transformer is currently partially implemented, but in a manner that potentially uses a great deal of memory (twice the memory needed for the geolocation arrays), and with still dubious correctness, but once approved this will be fixup up to at least be correct, though likely not efficient for the time being.

The GDALGenImgProjTransformer will be upgraded to instantiate the GeoLoc transformer (instead of an affine, gcp, or rpc transformer) if only geolocation information is available (done). However, the current GDALCreateGenImgProjTransformer() function does not provide a mechanism to select which transformation mechanism is used. So, for instance, if an affine transform is

available it will be used in preference to geolocation data. If bGCPUseOK is TRUE, gcps will be used in preference to geolocation data.

The gdalwarp program currently always sets bGCPUseOK to TRUE so there is no means for gdalwarp users select use of geolocation data in preference to gcps. Some modification to gdalwarp may be needed in the future in this regard.

2.5.5 Preserving Geolocation Through Translation

How do we preserve access to geolocation information when translating a dataset? Do applications like gdal_translate need special handling? Placement of the geolocation data in a special metadata domain means it won't be transferred in default translations.

2.6 RFC 5: Unicode support in GDAL

Author: Andrey Kiselev

Contact: dron@ak4719.spb.edu

Status: Development

2.6.1 Summary

This document contains proposal on how to make GDAL core locale independent preserving support for native character sets.

2.6.2 Main concepts

GDAL should be modified in a way to support three following main ideas:

1. Users work in localized environment using their native languages. That means we can not assume ASCII character set when working with string data passed to GDAL.
2. GDAL uses UTF-8 encoding internally when working with strings.
3. GDAL uses Unicode version of third-party APIs when it is possible.

So all strings, used in GDAL, are in UTF-8, not in plain ASCII. That means we should convert user's input from the local encoding to UTF-8 during interactive sessions. The opposite should be done for GDAL output. For example, when user passes a filename as a command-line parameter to GDAL utilities, that filename should be immediately converted to UTF-8 and only afetrwards passed to functions like GDALOpen() or OGROpen(). All functions, wich take character strings as parameters, assume UTF-8 (with except of several ones, which will do the conversion between different encodings, see **Implementation** (p. 19)). The same is valid for output functions. Output functions (CPLError/CPLDebug), embedded in GDAL, should convert all strings from UTF-8 to local encoding just before printing them. Custom error handlers should be aware of UTF-8 issue and do the proper transformation of strings passed to them.

The string encoding pops up again when GDAL needs to call the third-party API. UTF-8 should be converted to encoding suitable for that API. In particular, that means we should convert UTF-8 to UTF-16 before calling CreateFile() function in Windows implementation of VSIFOpenL(). Another example is a PostgreSQL API. PostgreSQL stores strings in UTF-8 encoding internally, so we should notify server that passed string is already in UTF-8 and it will be stored as is without any conversions and losses.

For file format drivers the string representation should be worked out on per-driver basis. Not all file formats support non-ASCII characters. For example, various .HDR labeled rasters are just 7-bit ASCII text files and it is not a good idea to write 8-bit strings in such a files. When we need to pass strings, extracted from such file outside the driver (e.g., in SetMetadata() call), we should convert them to UTF-8. If you just want to use extracted strings internally in driver, there is no need in any conversions.

In some cases the file encoding can differ from the local system encoding and we do not have a way to know the file encoding other than ask a user (for example, imagine a case when someone added a 8-bit non-ASCII string field to mentioned above plain text .HDR file). That means we can't use conversion form the local encoding to UTF-8, but from the file encoding to UTF-8. So we need a way to get file encoding in some way on per datasource basis. The natural solution of the problem is to introduce optional open parameter "ENCODING" to GDALOpen/OGROpen

functions. Unfortunately, those functions do not accept options. That should be introduced in another RFC. Fortunately, tehre is no need to add encoding parameter immediately, because it is independent from the general i18n process. We can add UTF-8 support as it is defined in this RFC and add support for forcing per-datasource encoding later, when the open options will be introduced.

2.6.3 Implementation

- New character conversion functions will be introduced in CPLString class. Objects of that class always contain UTF-8 string internally.

```
// Get string in local encoding from the internal UTF-8 encoded string.
// Out-of-range characters replaced with '?' in output string.
// nEncoding A codename of encoding. If 0 the local system
// encoding will be used.
char* CPLString::recode( int nEncoding = 0 );

// Construct UTF-8 string object from string in other encoding
// nEncoding A codename of encoding. If 0 the local system
// encoding will be used.
CPLString::CPLString( const char*, int nEncoding );

// Construct UTF-8 string object from array of wchar_t elements.
// Source encoding is system specific.
CPLString::CPLString( wchar_t* );

// Get string from UTF-8 encoding into array of wchar_t elements.
// Destination encoding is system specific.
operator wchar_t* (void) const;
```

- In order to use non-ASCII characters in user input every application should call setlocale(LC_ALL, "") function right after the entry point.
- Code example. Let's look how the gdal utilities and core code should be changed in regard to Unicode.

  For input instead of

```
pszFilename = argv[i];
if( pszFilename )
hDataset = GDALOpen( pszFilename, GA_ReadOnly );
```

  we should do

```
CPLString oFilename(argv[i], 0); // <-- Conversion from local encoding to UTF-8
hDataset = GDALOpen( oFilename.c_str(), GA_ReadOnly );
```

For output instead of

```
printf( "Description = %s\n", GDALGetDescription(hBand) );
```

we should do

```
CPLString oDescription( GDALGetDescription(hBand) );
printf( "Description = %s\n", oDescription.recode( 0 ) ); // <-- Conversion
// from UTF-8 to local
```

The filename passed to GDALOpen() in UTF-8 encoding in the code snippet above will be further processed in the GDAL core. On Windows instead of

```
hFile = CreateFile( pszFilename, dwDesiredAccess,
FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, dwCreationDisposition,
dwFlagsAndAttributes, NULL );
```

we do

```
CPLString oFilename( pszFilename );
// I am prefer call the wide character version explicitly
// rather than specify _UNICODE switch.
hFile = CreateFileW( (wchar_t *)oFilename, dwDesiredAccess,
        FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
        dwCreationDisposition,  dwFlagsAndAttributes, NULL );
```

- The actual implementation of the character conversion functions does not specified in this document yet. It needs additional discussion. The main problem is that we need not only local<->UTF-8 encoding conversions, but **arbitrary**<->UTF-8 ones. That requires significant support on software part.

2.6.4 Backward Compatibility

The GDAL/OGR backward compatibility will be broken by this new functionality in the way how 8-bit characters handled. Before users may rely on that all 8-bit character strings will be passed throgh the GDAL/OGR without change and will contain exact the same data all the way. Now it is only true for 7-bit ASCII and 8-bit UTF-8 encoded strings. Note, that if you used only ASCII subset with GDAL, you are not affected by these changes.

2.6.5 References

- FAQ on how to use Unicode in software:

  `http://www.cl.cam.ac.uk/~mgk25/unicode.html`

- FLTK implementation of string conversion functions:

  `http://svn.easysw.com/public/fltk/fltk/trunk/src/utf.c`
  `http://www.easysw.com/~mike/fltk/doc-2.0/html/utf_8h.html`

2.7 RFC 6: Geometry and Feature Style as OGR Special Fields

Author: Tamas Szekeres

Contact: szekerest@gmail.com

Status: Adopted

2.7.1 Summary

This proposal addresses and issue have been discovered long ago, and OGR provides no equivalent solution so far.

Some of the supported formats like Mapinfo.tab may contain multiple geometry types and style information. In order to hanlde this kind of data sources properly a support for selecting the layers by geometry type or by the style info would be highly required. For more details see the following MapServer related bugs later in this document.

All of the proposed changes can be found at the tracking bug of this RFC referenced later in this document.

2.7.2 Main concepts

The most reasonable way to support this feature is to extend the currently existing 'special field' approach to allow specifying more than one fields. Along with the already definied 'FID' field we will add the following ones:

- 'OGR_GEOMETRY' containing the geometry type like 'POINT' or 'POLYGON'.
- 'OGR_STYLE' containing the style string.
- 'OGR_GEOM_WKT' containing the full WKT of the geometry.

By providing the aforementioned fields one can make for example the following selections:

- select FID, OGR_GEOMETRY, OGR_STYLE, OGR_GEOM_WKT, * from MyTable where OGR_GEOMETRY='POINT' OR OGR_GEOMETRY='POLYGON'
- select FID, OGR_GEOMETRY, OGR_STYLE, OGR_GEOM_WKT, * from MyTable where OGR_STYLE LIKE 'BRUSH'
- select FID, OGR_GEOMETRY, OGR_STYLE, OGR_GEOM_WKT, * from MyTable where OGR_GEOM_WKT LIKE 'POLYGON'
- select distinct OGR_GEOMETRY from MyTable order by OGR_GEOMETRY desc

2.7.3 Implementation

There are two distinct areas where this feature plays a role

- Feature query implemented at ogrfeaturequery.cpp
- SQL based selection implemented at ogr_gensql.cpp and ogrdatasource.cpp

To specify arbitrary number of special fields we will declare an array for the field names and types in ogrfeaturequery.cpp as

```
char* SpecialFieldNames[SPECIAL_FIELD_COUNT]
    = {"FID", "OGR_GEOMETRY", "OGR_STYLE", "OGR_GEOM_WKT"};
swq_field_type SpecialFieldTypes[SPECIAL_FIELD_COUNT]
    = {SWQ_INTEGER, SWQ_STRING, SWQ_STRING, SWQ_STRING};
```

So as to make this array accessible to the other files the followings will be added to ogr_p.h

```
CPL_C_START
include "swq.h"
CPL_C_END

define SPF_FID 0
define SPF_OGR_GEOMETRY 1
define SPF_OGR_STYLE 2
define SPF_OGR_GEOM_WKT 3
define SPECIAL_FIELD_COUNT 4

extern char* SpecialFieldNames[SPECIAL_FIELD_COUNT];
extern swq_field_type SpecialFieldTypes[SPECIAL_FIELD_COUNT];
```

In ogrfeature.cpp the field accessor functions (GetFieldAsString, GetFieldAsInteger, GetFieldAsDouble) will be modified providing the values of the special fields by the field index

The following code will be added to the beginning of OGRFeature::GetFieldAsInteger:

```
int iSpecialField = iField - poDefn->GetFieldCount();
if (iSpecialField >= 0)
{
// special field value accessors
    switch (iSpecialField)
    {
    case SPF_FID:
        return GetFID();
    default:
        return 0;
    }
}
```

The following code will be added to the beginning of OGRFeature::GetFieldAsDouble:

```
int iSpecialField = iField - poDefn->GetFieldCount();
if (iSpecialField >= 0)
{
// special field value accessors
    switch (iSpecialField)
    {
    case SPF_FID:
        return GetFID();
    default:
        return 0.0;
    }
}
```

The following code will be added to the beginning of OGRFeature::GetFieldAsString:

```
int iSpecialField = iField - poDefn->GetFieldCount();
if (iSpecialField >= 0)
{
// special field value accessors
    switch (iSpecialField)
    {
    case SPF_FID:
        sprintf( szTempBuffer, "%d", GetFID() );
        return m_pszTmpFieldValue = CPLStrdup( szTempBuffer );
    case SPF_OGR_GEOMETRY:
        return poGeometry->getGeometryName();
    case SPF_OGR_STYLE:
        return GetStyleString();
    case SPF_OGR_GEOM_WKT:
        {
            if (poGeometry->exportToWkt( &m_pszTmpFieldValue ) == OGRERR_NONE )
                return m_pszTmpFieldValue;
            else
                return "";
        }
    default:
        return "";
    }
}
```

The current implementation of OGRFeature::GetFieldAsString uses a static string to hold the const char* return value that is highly avoidable and makes the code thread unsafe. In this regard the 'static char szTempBuffer[80]' will be changed to non static and a new member will be added to OGRFeature in ogrfeature.h as:

```
char * m_pszTmpFieldValue;
```

This member will be initialized to NULL at the constructor, and will be freed using CPLFree() at the destructor of OGRFeature.

In OGRFeature::GetFieldAsString all of the occurrences of 'return szTempBuffer;' will be changed to 'return m_pszTmpFieldValue = CPLStrdup(szTempBuffer);'

OGRFeature::GetFieldAsString is responsible to destroy the old value of m_pszTmpFieldValue at the beginning of the function:

```
CPLFree(m_pszTmpFieldValue);
m_pszTmpFieldValue = NULL;
```

In ogrfeaturequery.cpp we should change OGRFeatureQuery::Compile to add the special fields like:

```
iField = 0;
while (iField < SPECIAL_FIELD_COUNT)
{
    papszFieldNames[poDefn->GetFieldCount() + iField] = SpecialFieldNames[iField];
    paeFieldTypes[poDefn->GetFieldCount() + iField] = SpecialFieldTypes[iField];
    ++iField;
}
```

In ogrfeaturequery.cpp OGRFeatureQueryEvaluator() should be modifyed according to the field specific actions like

```
int iSpecialField = op->field_index - poFeature->GetDefnRef()->GetFieldCount();
if( iSpecialField >= 0 )
{
    if ( iSpecialField < SPECIAL_FIELD_COUNT )
    {
        switch ( SpecialFieldTypes[iSpecialField] )
        {
        case SWQ_INTEGER:
            sField.Integer = poFeature->GetFieldAsInteger( op->field_index );
        case SWQ_STRING:
            sField.String = (char*) poFeature->GetFieldAsString( op->field_index );
        }
    }
    else
    {
        CPLDebug( "OGRFeatureQuery", "Illegal special field index.");
        return FALSE;
```

```
    }
    psField =
}
else
    psField = poFeature->GetRawFieldRef( op->field_index );
```

In ogrfeaturequery.cpp OGRFeatureQuery::FieldCollector should be modifyed to add the field names like:

```
if( op->field_index >= poTargetDefn->GetFieldCount()
        && op->field_index < poTargetDefn->GetFieldCount() + SPECIAL_FIELD_COUNT)
        pszFieldName = SpecialFieldNames[op->field_index];
```

In ogrdatasource.cpp ExecuteSQL() will allocate the arrays according to the number of the special fields:

```
sFieldList.names = (char **)
        CPLMalloc( sizeof(char *) * (nFieldCount+SPECIAL_FIELD_COUNT) );
sFieldList.types = (swq_field_type *)
        CPLMalloc( sizeof(swq_field_type) * (nFieldCount+SPECIAL_FIELD_COUNT) );
sFieldList.table_ids = (int *)
        CPLMalloc( sizeof(int) * (nFieldCount+SPECIAL_FIELD_COUNT) );
sFieldList.ids = (int *)
        CPLMalloc( sizeof(int) * (nFieldCount+SPECIAL_FIELD_COUNT) );
```

And the fields will be added as

```
for (iField = 0; iField < SPECIAL_FIELD_COUNT; iField++)
{
    sFieldList.names[sFieldList.count] = SpecialFieldNames[iField];
    sFieldList.types[sFieldList.count] = SpecialFieldTypes[iField];
    sFieldList.table_ids[sFieldList.count] = 0;
    sFieldList.ids[sFieldList.count] = nFIDIndex + iField;
    sFieldList.count++;
}
```

For supporting the SQL based queries we should also modify the constructor of OGRGenSQLResultsLayer in ogr_gensql.cpp and set the field type properly:

```
else if ( psColDef->field_index >= iFIDFieldIndex )
{
    switch ( SpecialFieldTypes[psColDef->field_index - iFIDFieldIndex] )
    {
    case SWQ_INTEGER:
        oFDefn.SetType( OFTInteger );
        break;
    case SWQ_STRING:
        oFDefn.SetType( OFTString );
        break;
    case SWQ_FLOAT:
        oFDefn.SetType( OFTReal );
        break;
    }
}
```

Some of the queries will require to modify OGRGenSQLResultsLayer::PrepareSummary in ogr_gensql.cpp will be simplified (GetFieldAsString will be used in all cases to access the field values):

```
pszError = swq_select_summarize( psSelectInfo, iField,
poSrcFeature->GetFieldAsString( psColDef->field_index ) );
```

OGRGenSQLResultsLayer::TranslateFeature should also be modifyed when copying the fields from primary record to the destination feature

```
 if ( psColDef->field_index >= iFIDFieldIndex &&
            psColDef->field_index < iFIDFieldIndex + SPECIAL_FIELD_COUNT )
{
    switch (SpecialFieldTypes[psColDef->field_index - iFIDFieldIndex])
    {
    case SWQ_INTEGER:
        poDstFeat->SetField( iField, poSrcFeat->GetFieldAsInteger(psColDef->field_index) );
    case SWQ_STRING:
        poDstFeat->SetField( iField, poSrcFeat->GetFieldAsString(psColDef->field_index) );
    }
}
```

For supporting the 'order by' queries we should also modify OGRGenSQLResultsLayer::CreateOrderByIndex() as:

```
if ( psKeyDef->field_index >= iFIDFieldIndex)
{
    if ( psKeyDef->field_index < iFIDFieldIndex + SPECIAL_FIELD_COUNT )
    {
        switch (SpecialFieldTypes[psKeyDef->field_index - iFIDFieldIndex])
        {
        case SWQ_INTEGER:
            psDstField->Integer = poSrcFeat->GetFieldAsInteger(psKeyDef->field_index);
        case SWQ_STRING:
            psDstField->String = CPLStrdup( poSrcFeat->GetFieldAsString(psKeyDef->field_index) );
        }
    }
    continue;
}
```

All of the strings allocated previously should be deallocated later in the same function as:

```
if ( psKeyDef->field_index >= iFIDFieldIndex )
{
    /* warning: only special fields of type string should be deallocated */
    if (SpecialFieldTypes[psKeyDef->field_index - iFIDFieldIndex] == SWQ_STRING)
    {
        for( i = 0; i < nIndexSize; i++ )
        {
            OGRField *psField = pasIndexFields + iKey + i * nOrderItems;
            CPLFree( psField->String );
        }
    }
    continue;
}
```

When ordering by the field values the OGRGenSQLResultsLayer::Compare should also be modifyed:

```
if( psKeyDef->field_index >= iFIDFieldIndex )
    poFDefn = NULL;
else
    poFDefn = poSrcLayer->GetLayerDefn()->GetFieldDefn(
        psKeyDef->field_index );

if( (pasFirstTuple[iKey].Set.nMarker1 == OGRUnsetMarker
        && pasFirstTuple[iKey].Set.nMarker2 == OGRUnsetMarker)
    || (pasSecondTuple[iKey].Set.nMarker1 == OGRUnsetMarker
        && pasSecondTuple[iKey].Set.nMarker2 == OGRUnsetMarker) )
    nResult = 0;
else if ( poFDefn == NULL )
{
```

```
    switch (SpecialFieldTypes[psKeyDef->field_index - iFIDFieldIndex])
    {
    case SWQ_INTEGER:
        if( pasFirstTuple[iKey].Integer < pasSecondTuple[iKey].Integer )
            nResult = -1;
        else if( pasFirstTuple[iKey].Integer > pasSecondTuple[iKey].Integer )
            nResult = 1;
        break;
    case SWQ_STRING:
        nResult = strcmp(pasFirstTuple[iKey].String,
                         pasSecondTuple[iKey].String);
        break;
    }
}
```

2.7.4 Adding New Special Fields

Adding a new special field in a subsequent development phase is fairly straightforward and the following steps should be made:

1. In ogr_p.h a new constant should be added with the value of the SPECIAL_FIELD_COUNT and SPECIAL_FIELD_COUNT should be incremented by one.

2. In ogrfeaturequery.cpp the special field string and the type should be added to SpecialFieldNames and SpecialFieldTypes respectively

3. The field value accessors (OGRFeature::GetFieldAsString, OGRFeature::GetFieldAsInteger, OGRFeature::GetFieldAsDouble) should be modifyed to provide the value of the new special field. All of these functions provide const return values so GetFieldAsString should retain the value in the m_pszTmpFieldValue member.

4. When adding a new value with a type other than SWQ_INTEGER and SWQ_STRING the following functions might also be modified accordingly:

- OGRGenSQLResultsLayer::OGRGenSQLResultsLayer
- OGRGenSQLResultsLayer::TranslateFeature
- OGRGenSQLResultsLayer::CreateOrderByIndex
- OGRGenSQLResultsLayer::Compare
- OGRFeatureQueryEvaluator

2.7.5 Backward Compatibility

In most cases the backward compatibility of the OGR library will be retained. However the special fields will potentially conflict with regard fields with the given names. When accessing the field values the special fields will take pecedence over the other fields with the same names.

When using OGRFeature::GetFieldAsString the returned value will be stored as a member variable instead of a static variable. The string will be deallocated and will no longer be usable after the destruction of the feature.

2.7.6 Regression Testing

A new gdalautotest/ogr/ogr_sqlspecials.py script to test support for all special fields in the ExecuteSQL() call and with WHERE clauses.

2.7.7 Documentation

The OGR SQL document will be updated to reflect the support for special fields.

2.7.8 Implementation Staffing

Tamas Szekeres will implement the bulk of the RFC in time for GDAL/OGR 1.4.0.

Frank Warmerdam will consider how the backward compatibility issues (with special regard to the modified lifespan of the GetFieldAsString returned value) will affect the other parts of the OGR project and will write the Python regression testing script.

2.7.9 References

- Tracking bug for this feature (containing all of the proposed code changes):
  `http://bugzilla.remotesensing.org/show_bug.cgi?id=1333`
- MapServer related bugs:
  `http://mapserver.gis.umn.edu/bugs/show_bug.cgi?id=1129`
  `http://mapserver.gis.umn.edu/bugs/show_bug.cgi?id=1438`

2.7.10 Voting History

Frank Warmerdam +1 Daniel Morissette +1 Howard Butler +0 Andrey Kiselev +1

2.8 RFC 7: Use VSILFILE for VSI*L Functions

Author: Eric Dönges

Contact: Eric.Doenges@gmx.net

Status: Proposed

2.8.1 Purpose

To change the API for the VSI*L family of functions to use a new data-type VSILFILE instead of the current FILE.

2.8.2 Background, Rationale

Currently, GDAL offers two APIs to abstract file access functions (referred to as VSI* and VSI*L in this document). Both APIs claim to operate on FILE pointers; however, the VSI*L functions can only operate on FILE pointers created by the VSIFOpenL function. This is because VSIFOpenL returns a pointer to an internal C++ class typecast to a FILE pointer, not an actual FILE pointer. This makes it impossible for the compiler to warn when the VSI* and VSI*L functions are inappropriately mixed.

2.8.3 Proposed Fix

A new opaque data-type VSILFILE shall be declared. All VSI*L functions shall be changed to use this new type instead of FILE. Additionally, any GDAL code that uses the VSI*L functions must be changed to use this data-type as well.

2.8.4 Compatibility Issues, Transition timeline

In order to allow the compiler to detect inappropriate parameters passed to any of the VSI*L functions, VSILFILE would have to be declared with the help of an empty forward declaration, i.e.

```
typedef struct VSILFILE VSILFILE
```

with the struct VSILFILE itself left undefined. However, this would break source compatibility for any existing code using the VSI*L API.

Therefore, VSILFILE* will be declared as a void pointer for now, and the change to a distinct pointer type deferred to a future release of gdal that is not constrained with backward compatibility issues. While this will not solve the primary issue (no warnings/errors from the compiler), looking at the declarations of the VSI*L functions will at least make it immediately clear that these functions cannot be expected to work if passed a FILE pointer.

2.9 RFC 8: Developer Guidelines

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: draft

2.9.1 Purpose

This document is intended to document developer practices for the GDAL/OGR project. It will be an evolving document.

2.9.2 Portability

GDAL strives to be widely portable to 32bit and 64bit computing environments. It accomplishes this in a number of ways - avoid compiler specific directives, avoiding new, but perhaps not widely available aspects of C++, and most importantly by abstracting platform specific operations in CPL functions in the gdal/port directory.

Generally speaking, where available CPL functions should be used in preference to operating system functions for operations like memory allocation, path parsing, filesystem io, multithreading functions, and ODBC access.

2.9.3 Variable Naming

Much of the existing GDAL and OGR code uses an adapted Hungarian naming convention. Use of this convention is not manditory, but when maintaining code using this convention it is desirable to continue adhering to it with changes. Most importantly, please avoiding using it improperly as that can be very confusing.

In Hungarian prefixing the prefix tells something about about the type, and potentially semantics of a variable. The following are some prefixes used in GDAL/OGR.

- **p**: pointer
- **a**: array
- **sz**: zero terminated string (eg." char szName[100];")
- **psz**: pointer to a zero terminated string. (eg. "char *pszName;")
- **n**: integer number (size unspecified)
- **i**: integer number used as a zero based array or loop index.
- **f**: floating point value (single precision)
- **df**: floating point value (double precision)
- **o**: c++ object
- **os**: CPLString
- **h**: an opaque handle (such as GDALDatasetH).

Prefix can be stacked. The following are some examples of meaningful variables.

- **char** ∗∗**papszTokens**: Pointer to an array of strings.
- **int** ∗**panBands**: Pointer to an array of numbers.
- **double** ∗**padfScanline**: Pointer to an array of doubles.
- **double** ∗**pdfMeanRet**: Pointer to a single double.
- **GDALRasterBand** ∗**poBand**: Pointer to a single object.

It may also be noted that the standard convention for variable names is to capitalize each word in a variable name.

2.9.4 Headers, and Comment Blocks

2.9.5 Misc. Notes

- Use lower case filenames.
- Use .cpp extension for C++ files (not .cc).
- Avoid spaces or other special characters in file or directory names.
- Use 4 character indentation levels.
- Use spaces instead of hard tab characters in source code.
- Try to keep lines to 79 characters or less.

2.10 RFC 9: GDAL Paid Maintainer Guidelines

Author: Frank Warmerdam

Contact: warmerdam@pobox.com

Status: Proposed

2.10.1 Purpose

To formalize guidelines for the work of maintainers paid out of GDAL project sponsorship funds.

2.10.2 Responsibilities

1. Analyse and where possible fix bugs reported against GDAL.
2. Run, review and extend the test suite (via buildbot, etc).
3. Maintain and extend documentation.
4. Assist integrating new contributed features.
5. Help maintain project infrastructure (mailing lists, buildbot, source control, etc)
6. Provide user support on the project mailing lists, and in other venues.
7. Develop new capabilities.

Bug fixing and maintenance should be focused on GDAL/OGR, but as needed will extend into sub-projects such as libtiff, libgeotiff, Shapelib and MITAB as long it is to serve a need of the GDAL/OGR project.

In order to provide reasonable response times the maintainer is expected spend some time each week addressing new bugs and user support. If the maintainer will be unavailable for an extended period of time (vacation, etc) then the supervisor should be notified.

2.10.3 Direction

The maintainer is generally subject to the project PSC. However, for day to day decisions one PSC member will be designated as the supervisor for the maintainer. This supervisor will prioritize work via email, bug assignments, and IRC discussions.

The supervisor will try to keep the following in mind when prioritizing tasks.

- Bug reports, and support needs of Sponsors should be given higher priority than other tasks.
- Areas of focus identified by the PSC (ie. multi-threading, SWIG scripting) should be given higher priority than other tasks.
- Bugs or needs that affect many users should have higher priority.
- The maintainer should be used to take care of work that no one else is willing and able to do (ie. fill the holes, rather than displacing volunteers)
- Try to avoid tieing up the maintainer on one big task for many weeks unless directed by the PSC.

- The maintainer should not be directed to do work for which someone else is getting paid.

Substantial new development projects will only be taken on by the maintainer with the direction of a PSC motion (or possibly an RFC designating the maintainer to work on a change).

Note that the maintainer and the maintainer supervisor are subject to the normal RFC process for any substantial change to GDAL.

2.10.4 Reporting

The maintainer will produce a brief bi-weekly report to the gdal-dev list indicating tasks worked on, and a more detailed timesheet for the supervisor.

This is intended to provide visibility into status, accomplishments, and time allocation. It also gives an opportunity for the PSC to request a "course correction" fairly promptly.

2.11 RFC List

A list of all GDAL/OGR RFC documents, with status.

- **RFC 1: Project Management Committee Guidelines** (p. 4) (Adopted)
- **RFC 2: Migration to OSGeo Subversion Repository** (p. 6) (Implementation)
- **RFC 3: GDAL Commiter Guidelines** (p. 8) (Adopted)
- **RFC 4: Geolocation Arrays** (p. 12) (Development)
- **RFC 5: Unicode support in GDAL** (p. 15) (Development)
- **RFC 6: Geometry and Feature Style as OGR Special Fields** (p. 19) (Adopted)
- **RFC 7: Use VSILFILE for VSI∗L Functions** (p. 28) (Proposed)
- **RFC 8: Developer Guidelines** (p. 29) (Development)
- **RFC 9: GDAL Paid Maintainer Guidelines** (p. 31) (Proposed)