



# OSGeo Journal Volume 8

## February 2011

FOSS4G 2009  
Conference Proceedings

OSGeo Community  
News & Announcements  
Case Studies  
Integration Examples



**FOSS4G**

D E N V E R **2011**

SEPTEMBER 12-16

The Annual International

FREE & OPEN SOURCE  
SOFTWARE FOR GEOSPATIAL

Conference Event

2011.FOSS4G.ORG



## Volume 8 Contents

<b>Editorial</b>	<b>2</b>
From the Editor . . . . .	2
<b>News &amp; Announcements</b>	<b>3</b>
Brief News and Event Announcements from the OSGeo Community . . . . .	3
<i>r.in.swisstopo</i> . . . . .	5
<b>Case Studies</b>	<b>8</b>
An Image Request Application Using FOSS4G Tools	8

<b>Integration Examples</b>	<b>10</b>
Exporting Geospatial Data to Web Tiled Map Ser- vices using GRASS GIS . . . . .	10
<b>FOSS4G 2009 Conference Proceedings</b>	<b>15</b>
From the Academic Track Chair . . . . .	15
Geoprocessing in the Clouds . . . . .	17
Media Mapping . . . . .	23
MapWindow 6.0 . . . . .	31
A Data System for Visualizing 4-D Atmospheric CO2 Models and Data . . . . .	37
Collaborative Web-Based Mapping of Real-Time Flight Simulator and Sensor Data . . . . .	48
A Modular Spatial Modeling Environment for GIS	53

## From the Editor

OSGeo has just past its 5th birthday, along with this 8th volume of the OSGeo Journal! With this edition we bring a few news headlines from the past couple months, a few general articles and, most significantly, several top papers from the **FOSS4G 2009** conference event held in Sydney, Australia.



The Journal has become a diverse platform for several groups and growth in each area is expected to continue. The key groups that read and contribute to the Journal include software developers sharing information about their projects or communities, power users showing off their solutions, academia seeking to publish their research and observations in a peer-reviewed, open source friendly medium. OSGeo also uses the Journal to share community updates and the annual reports of the organisation.

Welcome to those of you who are new to the OSGeo Journal. Our Journal team and volunteer reviewers and editors hope you enjoy this volume. We also invite you to submit your own articles to any of our various sec-

tions. To submit an article, register as an "author" and sign in at <http://osgeo.org/ojs>. Then when you log in you will see an option to submit an article.<sup>1</sup>

We look forward to working with, and for, you in the upcoming year. It's sure to be an interesting year as we see OSGeo, Open Source in general and all our relate communities continue to grow. Nowhere else is this growth more apparent than at our annual conference: **FOSS4G 2011 Denver**, September, 2011.<sup>2</sup> Keep an eye on your OSGeo mailing lists, blogs and other feeds to follow the latest FOSS4G announcements, including the invitation to submit presentation proposals.<sup>3</sup> It will be as competitive as ever to get a speaking slot, so be sure to make your title and abstract really stand out.

Wishing you the best for 2011 and hoping to see you in Denver!

Tyler Mitchell  
[tmitchell@osgeo.org](mailto:tmitchell@osgeo.org)  
 Editor in chief, OSGeo Journal  
 Executive Director, OSGeo

<sup>1</sup>The direct URL for article submission is: <https://www.osgeo.org/ojs/index.php/journal/author/submit>

<sup>2</sup>FOSS4G 2011 Denver: <http://2011.foss4g.org>

<sup>3</sup>FOSS4G 2011 Abstract Submission: <http://2011.foss4g.org/program>

---

# FOSS4G 2009 Conference Proceedings

---

## From the Academic Track Chair

*Prof. Thierry Badard*

The FOSS4G 2009 academic track aimed to bring together researchers, developers, users and practitioners – all who were carrying out research and development in the free and open source geospatial fields and who were willing to share original, recent developments and experiences.



The primary goal was to promote cooperative research between OSGeo developers and academia, but the academic track has also acted as an inventory of current research topics. This track was the right forum to highlight the most important research challenges and trends in the domain and let them become the basis for an informal OSGeo research agenda. It has fostered interdisciplinary discussions in all aspects of the free and open source geospatial domains. It was organized to promote networking between the participants, to initiate and favour discussions regarding cutting-edge technologies in the field, to exchange research ideas and to promote international collaboration.

In addition to the OSGeo Foundation<sup>23</sup>, the ICA (International Cartographic Association) working group on open source geospatial technologies<sup>24</sup> was proud to support the organisation of the track.

The coordinators sought to gather paper submissions globally that addressed theoretical, technical, and practical topics related to the free and open source geospatial domain. Suggested topics included, but were not limited to, the following:

- State of the art developments in Open Source GIS
- Open Source GIS in Education
- Interoperability and standards - OGC, ISO/TC 211, Metadata
- Spatial Data Infrastructures and Service Oriented Architectures
- Free and open source Web Mapping, Web GIS and Web processing services
- Cartography and advanced styling
- Earth Observation and remote sensing
- Spatial and Spatio-temporal data, analysis and integration
- Free and Open Source GIS application use cases in Government, Participatory GIS, Location based services, Health, Energy, Water, Urban and Environmental Planning, Climate change, etc.

In response to the call for papers, 25 articles were submitted to the academic track. The submissions were highly diversified, and came from USA, Canada, Thailand, Japan, South Korea, Sri Lanka, Australia, New Zealand, Italy, Denmark, France, Germany, Switzerland, Romania and Turkey. Selection of submissions were based on the full papers received. All submissions were thoroughly peer reviewed by two to three members of the international scientific committee and refereed for their quality, originality and relevance. The scientific committee selected 12 papers (48% acceptance rate) for presentation at the FOSS4G 2009 conference. From those, 6 papers were accepted for presentation in the proceedings of the academic track, which are published in this volume of the OSGeo Journal. They correspond to the 6 best papers assessed by the international scientific committee.

The accepted and published papers covered a wide

<sup>23</sup>OSGeo: Open Source Geospatial Foundation: <http://osgeo.org>

<sup>24</sup>ICA open source working group: <http://ica-opensource.scg.ulaval.ca/>



range of cutting-edge research topics and novel applications on Free and Open Source Geospatial technologies. I am particularly proud and happy to see some very high quality scientific contributions published in the OSGeo Journal. This will undoubtedly encourage more interesting research to be published in this volume, as our OSGeo journal is an open access journal. In addition, it helps draw attention to this important project of the OSGeo Foundation. I hope the publication of these proceedings in the OSGeo journal will encourage future scientists, researchers and members of academia to consider the OSGeo Journal as an increasingly valuable place to publish their research works and case studies.

As a concluding note, I would like to take the opportunity to thank the individuals and institutions that made the FOSS4G 2009 academic track possible. First,

I would like to thank the international scientific committee members and external reviewers for evaluating the assigned papers in a timely and professional manner. Next, I would like to recognize the tremendous efforts put forward by members of the local organizing committee of FOSS4G 2009 for accommodating and supporting the academic track. Finally, I want to thank the authors for their contributions, efforts, patience and support that made this academic track a huge success.

*January, 2011*

*Prof. Thierry Badard*

*Laval University, Canada*

*Chair, FOSS4G 2009 Academic Track*

*Co-chair, ICA Working Group on Open Source Geospatial Technologies*

# MapWindow 6.0

## An Extensible Architecture for Cartographic Symbolology

*Harold A. Dunsford Jr., Daniel P. Ames*

### Abstract

A robust, extensible architecture is critical to open source projects that have a distributed developer and user base. The MapWindow 6.0 project is using a new architectural paradigm where extensibility is handled from several different plug-in points, rather than a single, application wide design. This allows new kinds of extensibility to be explored such as tools and data providers in addition to the more conventional application wide extensibility. This presentation outlines some of the improvements in the built in cartography, but primarily addresses the .Net architectural decisions that permit run-time discovery of new kinds of custom symbology. Improvements include layering of different kinds of symbols to make a compound symbol as well as establishing cartographic sub-categories based on vector attributes or raster values. The open ended framework allows for an extremely flexible system of run-time discovery so that the core libraries do not have to be recompiled each time an external cartographic improvement is developed.

### Introduction

The problem addressed in this paper is the inability for an open source core application to anticipate all of the symbolic requirements for new sorts of data. Changes in the design model for the 6.0 version of the open source MapWindow GIS project allow for new kinds of plug-ins. One new kind of plug-in actually allows for external libraries to control the business logic of data management for a specific data format. The software that controls the business logic that runs the interface based run-time recognition of these new data providers has also been encapsulated in the form of a non-graphical component that can be easily added to a new project as easily as dragging the Map control or the legend onto that project. The inevitable consequence for this is that eventually there may be new styles of data which need to be symbolized in an unconventional way. This paper seeks to address the problem of how we can design an architecture that is at once extremely flexible and versatile, supplying a built in symbol set that is as rich as professional software, but that is also malleable, so that future developers can easily extend the symbolic capabilities without having to recompile the architectural core.

The techniques outlined in this paper are important

because architectures that support extensibility form a robust framework for successful open source GIS platforms to build on. While this quality is important for both proprietary and open source systems, it is essential when a spatially distributed developer base must coordinate their efforts. Multi-tiered, modular and transparent design standards allow for greater security and design control of the low-level, shared libraries, and also act as a contract to unify a wide range of extensibility and customization that is added on top of that core. This provides future and co-developers with a common platform that can be extended without fear of breaking other parts of the code – hence saving time and development costs.

We propose that it is non-obvious as to how to use a common interface or custom attribute to allow the core library and other developers to use extension classes effectively if the most critical content is not described by the interface itself. The conventional, run-time discovery of extension classes traditionally relies on using either System.Reflection or the Microsoft Add-In framework in order to specify light weight contract interfaces that then can be fleshed out with code that implements those interfaces. For something like a data provider, where the end result is an in-memory data format that always matches a given interface, this sort of traditional extensibility interface is ideal because you know what to expect from each additional data provider. Symbology, however, is something that should support a rich and versatile collection of properties that can't be predicted in advance. For instance, extending a point symbol to be represented by an image requires completely different attributes and properties than describing the point using a color, size and shape. What's wrong with previous proposed solutions?

Previous GIS architectures fail to address these versatile areas of extensibility. The conventional open source approach is that tasks like handling data formats and rendering those data formats are handled internally and modified or developed by only a select few that are privileged to be working on the core library. Even the most cutting edge advancements in Microsoft's Add-In Framework simply allow contract interfaces to be updated to newer versions, and don't address a way to easily extend the project with effectively unbounded components.

The first key component of our approach is the design of the base interface. This includes an *ISymbol*, for points, an *IStroke* for lines and an *IPattern* for polygons. These basic elements have minimal common attributes, but they all have methods that allow for those symbols to render themselves, given the set of coordinates, or a graphics path, to draw. The important thing from the standpoint of the rendering is that it won't matter

what kind of properties control that rendering method, since all the symbols, for instance, evoke the same basic drawing method. These base interfaces also allow for run-time identification of classes that extend symbology.

The second topic of discussion by this paper is how to design a corresponding user interface that can allow for largely unpredictable symbol elements to be successfully modified by the user without knowing them in advance. The Dot Net Property Dialog component will be discussed as an option for a default user interface if one is not specified, but more importantly we will illustrate how software can allow developers to provide a custom control, or tab-page where they handle their own symbology.

Finally, we will address using extension methods (first made available with the 3.0 version of the .Net Framework) to allow developers to create new methods that are programmatically discoverable to intellisense, but also allow easy configuration of their new symbols without requiring the core library to be recompiled.

## Background

### MapWindow

The MapWindow project was first established in 1998 at Utah Water Research Lab in Logan as an alternative to using MapObjects LT 1.0. (1) The proprietary controls provided by ESRI prevented users from modifying the underlying data, however, which was of limited use for research oriented applications. The project requirements included being able to dynamically alter the shapes of vector data, or access the data values for grids. They created the core MapWinGIS.ocx component, an ActiveX control that could provide the low level access to the data formats that developers could then rapidly turn into successful projects. This ultimately led to the development of the MapWindow application because many of the common features that were shared between projects ended up being replicated over and over again. The fully developed project today is called MapWindow 4.x.

The MapWindow 6.0 project is focused on developing modular components written in C#. (3) Since everything written for MapWindow 6.0 is in a managed, Dot-Net language, it will be far more portable in terms of using the project in web applications or across platforms using the open source Mono framework. The MapWindow 6.0 version of the project began in the summer of 2007 as an effort to develop a topology toolkit for MapWindow 4.x. In 2007, the team added the Net Topology Suite into the project. This required such a serious re-thinking of the underlying objects that it was beneficial to start development of a completely new version of MapWindow from scratch, which is now MapWindow version 6.0. (2)

## Other GIS Extensibility Architectures

### ArcGIS

The ESRI ArcGIS object model consists of a host of interconnected objects, each with a very precisely defined role. The paradigm is to provide programmatic, macro-style access to the underlying objects through interfaces that restrict the functionality. In the Visual Basic for Applications (VBA) Macro development environment that is associated with ArcGIS versions 8.0 and larger, directly accessing an object doesn't expose the majority of its properties or methods. An example is the MxDocument object. In order to access more, you must first dimension a new IMxDocument interface, and point it at the object. This allows tight control over what aspects of the software external developers can control by exposing only a limited subset of members. Access permissions could be adjusted so that the full object is only viewable to developers working within the ArcMap project itself. There is no model in place to allow other developers to provide the base application with support for new data formats, though they are beginning to rely on the open source community projects like GDAL for some of their data member support. They also do not provide direct access to outside developers to low level functionality like rendering.

### Grass

The extensibility model for the GRASS project involves creating new libraries that can perform new, independent operations. As is evident from their book *Open Source GIS A GRASS GIS Approach*, GRASS 6 is written in the ANSI C programming language and hosts more than 350 modules for management, processing, analysis and visualization of GIS data. The strategy that they have adopted is to require that all data formats be converted to their standardized raster and vector formats before any other module can work with the data. This allows for analysis of data directly from a file that might be too large to store in memory, while reducing the complexity of the analysis methods to working with a single data format. They also recognize that not every user will be an expert coder. In order to support this intermediate level of programmer, the GRASS supports script programming. UNIX Shell, PERL and Python scripts are supported, allowing repetitious tasks to be handled through their scripting language. They are the most mature version of truly open-source GIS today, and can be thought of as a textbook example of how to run a successful, long-term open source venture. GRASS shows us that in order to be a good, open-source project; you must have a framework that allows for future development and expansion over a long time. Our extensibility model explores how to give the .Net libraries a common GIS framework to use in order to talk to each other through the use of common interfaces, rather than the use of a common

data format, but ultimately the long term goal is the same.

### Quantum GIS

Given that there are hundreds of open source projects that feature GIS today, some of which are featured at <http://opensourcegis.org/>, it is hard to choose one to best illustrate the extensibility models that are currently available from applications that fall in the middle spectrum. One of the more complete and well known systems is Quantum GIS. The approach from Quantum GIS has always been more like a standard windows-style programming environment, instead of the traditional, Linux-style command prompt that was the principal mechanism for working with GRASS until just recently.

Quantum GIS supports a plug-in architecture that is more reminiscent of what you would likely find in a traditional software package. They even support a special type of Data Provider plug-in that allows developers to specifically extend the data formats that can be supported by the project. There are posts about Data Providers on their bulletin going back as far as 2006, so even though a big part of this presentation is about introducing plug-ins with distinct capabilities, this isn't an entirely unproven concept. The plug-in architecture for QGIS works by using python script or C++ files that satisfy certain criteria, i.e. hosting a particular script file with the name plugin.py and in essence writing script to match specific method names or schema. The weakness of this model is that the powerful development environment tools like intellisense are not generally available when working with a scripting language. With an interface, modern development environments are able to flesh out a skeleton with the correct signature that is type-checked at compile time. Further, while the special plug-ins exist to support data formats, those must first be accepted by the community and then manually added into the core library.

## Programming Methods

### Interfaces

An interface acts as a kind of skeleton framework for classes. In the C# language, dual-inheritance is not allowed, but it is possible to implement as many interfaces as you want. (5) Therefore, it has become fairly conventional to devise small interfaces that can be re-used in many different contexts. Many examples are built into the .Net Framework, such as the ICloneable interface which simply specifies that there will be a Clone() method that returns a new object. There are many classes that support this method, and those classes come from very diverse sets of class hierarchies. An interface can be thought of as a contract that designates what a particular class will do, regardless of

the actual code that is used to fulfill the contract. As was illustrated in an earlier paper published in Position IT (2) the minor performance characteristics of virtual calls through an interface is insignificant compared to the performance penalty of using property accessors. There is a limitation in the sense that public variables called fields cannot be defined on an interface, and so using an interface forces the use of property accessors or methods, and using property accessors can slow down performance significantly in large loops.

### Property-Grid

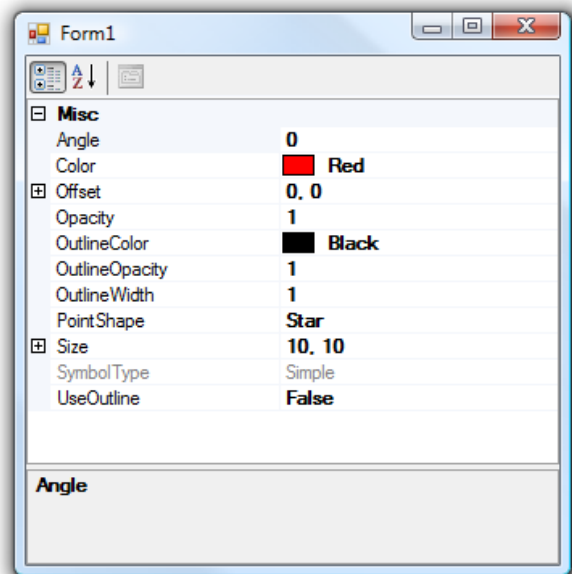


Figure 1: Property-Grid Control

The Property Grid control shown in Figure 1 is a .Net control that creates a kind of tabular layout that itemizes each of the public properties on a class, and next to that, provides an interactive spot where a value can be changed. The default behavior of the editing region is type dependant, so that simple values might only allow a text-box style editing, while more complex members might have a drop-down control or even provide a button that launches an entire dialog. These behaviors can be customized for new class data types using so called 'Editor' classes that control how the property grid behaves during the editing process. The editor to use for a new class is controlled by the use of attributes.

The significance of this control is that it does not require any pre-existing knowledge of what properties exist on a variable. While we don't recommend that everything relies on property grids, we are using it as an example of an open ended user interface design. Without relying on anything more complex than a .Net property grid, it then becomes possible to provide a default user interface for any new symbol classes that



do not explicitly define a user interface editor.

### Extension Methods

Extension methods have been a part of the of the .Net framework since version 3.0. These methods appear to extend the number and type of methods that can be accessed programmatically from an existing class or interface, even if the class is a sealed class and cannot be modified through inheritance. (4) The methods and properties that appear in intellisense are normally limited by the type definition of that variable. However, with extension methods, new methods can be 'appended' to the existing class, without ever modifying the source code for the class. In the visual studio development environment, these extension methods are designated by a purple down arrow to the left of the method. The importance of extension methods for this paper is that they represent the means by which intermediate level developers can design programmatic access to control new symbology members, making them programmatically discoverable through intellisense to future developers.

Between open ended user interface components like the property grid dialog and the extension methods, applications like MapWindow can support a new design concept which can easily build from or extend core libraries.

```

1 using System.Drawing;
2 namespace Examples
3 {
4     public static class PointEM
5     {
6         /// <summary>
7         /// This method transposes the X and the Y terms
8         /// </summary>
9         public static void Transpose(this Point self)
10        {
11            int temp = self.X;
12            self.Y = self.X;
13            self.X = temp;
14        }
15
16        public static void Test()
17        {
18            Point myPoint = new Point(3, 4);
19            myPoint.
20        }
21    }
22 }
23

```

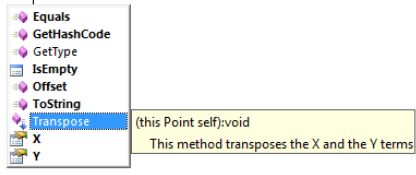


Figure 2: Transpose Extension Method

## MapWindow 6.0 Symbology

### Symbol Class Hierarchy

As of 6/29/2009, the architecture for the symbology interfaces follows a standard idea where a random but simple coloring scheme is applied to an entire layer as soon as the layer is added to the map. The layer does not host the descriptive characteristics directly, but rather stores a reference to a single scheme. The type of scheme depends on the type of features being repre-

sented. A PointScheme, for instance, works with point data, while PolygonScheme and LineScheme represent schemes that are specific to describing polygons and lines. The classes use a collection concept, so that each Scheme represents a collection of Categories. The strict role of a category is that it combines a query string with a Symbolizer. The string is used to select the members that the symbology will be applied to. This enables an easy programmatic access to hosting complex attribute based symbology. Finally, a PointSymbolizer is made up of at least one, but potentially several overlapping Symbols. A LineSymbolizer is made up of Strokes. A PolygonSymbolizer is made up of Patterns.

### Point Symbology

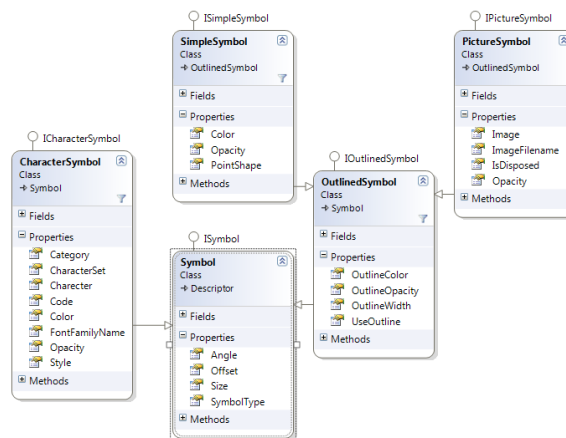
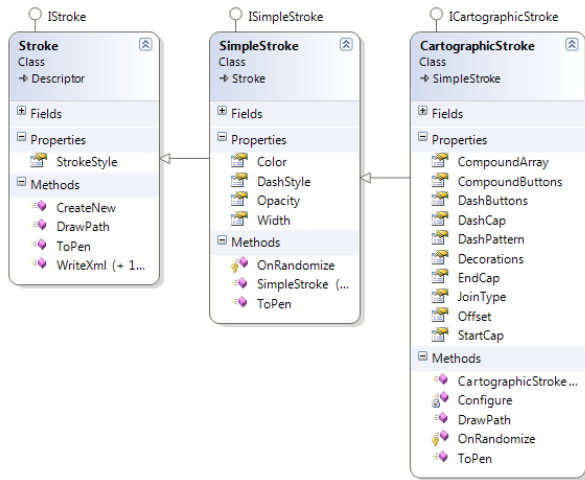


Figure 3: Point Symbol Classes

While each of the members above has a corresponding interface, it should be pointed out that only the lowest level need be developed in order for the developer to extend the graphical representations for vector features. For instance, if you wanted to design a new class to draw point types, implementing the ISymbol is sufficient, allowing it to use the pre-existing structure of symbolizers, categories and schemes. The existing symbol classes implement this interface and currently control their own drawing. All of the point symbol classes provide an offset, size and angle, but even characteristics like color are not universal. A PictureSymbol uses an image to control the drawing and has no information about coloring, though it can have an outline. A CharacterSymbol provides properties to control the font family name, the style and the character. The Characters can consist of artistically created symbol sets which support vector drawing and so will look good at any scale. A SimpleSymbol provides the most basic point symbology with an enumeration of default shapes. This simply uses GDI+ drawing methods to build the shapes programmatically.

**Line Symbolology**



**Figure 4: Line Stroke Classes**

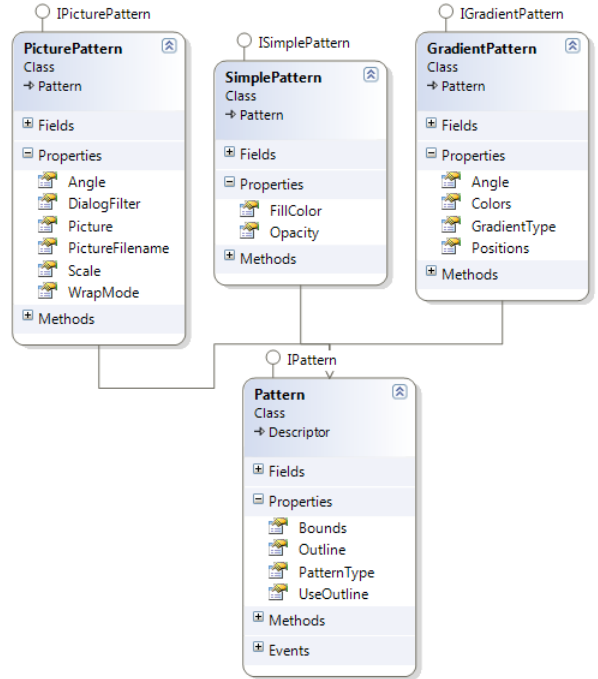
The equivalent concept for lines uses the IStroke interface. Each stroke represents a pass with a pen. At the moment, there is a SimpleStroke and CartographicStroke. The simple stroke allows for a color, width and dash pattern selected from an enumeration. It uses a default choice of rounded end caps. The Cartographic stroke extends the capabilities of the simple stroke. It adds a custom dash pattern, custom contour pattern, line cap styles, and the ability to specify a number of point symbols as decorations along the line. Since these strokes are then layered one on top of the other, very complex line structures are now possible. If, however, a user wanted to design a new kind of stroke that was designed entirely by point symbols drawn at fixed distances, it could be done easily by creating a plug-in that implements the IStroke interface. Currently, the drawing is handled by passing the GraphicsPath for the lines to draw (after they have been translated to screen coordinates) to each stroke in sequence where it handles its own drawing using GDI+ graphics calls.

**Polygon Symbolology**

The final set of descriptive symbols is for polygons. The PolygonSymbolizer is slightly more complex than the symbolizers for points or lines because in addition to having a collection of patterns, it also specifies a line symbolizer to use for drawing the borders. In this way, we get to re-use the drawing code for the lines when drawing the borders of the polygon. The individual patterns include a SimplePattern that only specifies a fill color. A PicturePattern is created using a collection of tiled images. Finally a GradientPattern uses linear, circular or rectangular gradients with various rotations and colors. The gradient can be controlled programmatically to work with many colors and positions (ranging from 0 to 1).

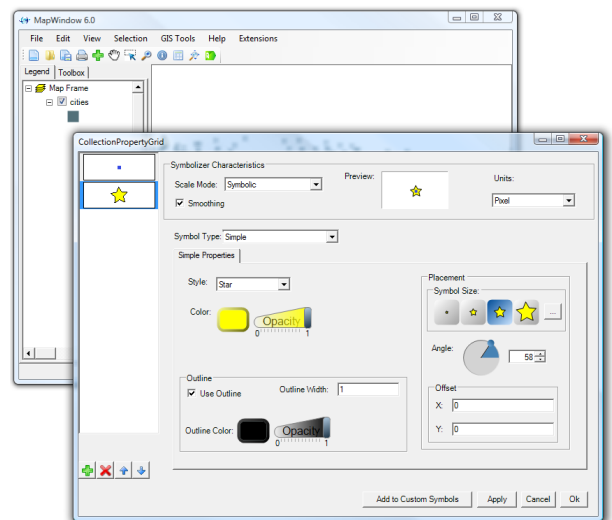
These new cartographic capabilities give an extraor-

dinarily rich way to draw, color, or depict vector feature content, but what is most innovative is that the symbology itself is extensible. Raster symbology also exists, but is not tremendously different from techniques dating back to earlier versions of MapWindow, where several color breaks subdivide a raster.



**Figure 5: Polygon Pattern Classes**

**User Interface Design**



**Figure 6: Point Symbolizer User Interface**

Double clicking on the representation below the layer automatically launches a complex dialog that allows the customization of that symbol. As is illustrated in Figure 3, a fairly straight forward collection of properties exists described on the Simple Properties tab. A Drop-down currently set to 'Simple' controls what elements appear in the tab control. The extensibility

allows new Symbol types to appear in this drop-down, and, while still in the planning stages, we plan on supporting a ISymbolUIEditor interface that allows users to specify the exact layout of a tab control for editing their custom symbol type chosen from the drop-down. In such a case, the layering (shown in the graphical list representations on the left side of the form) and content in the placement group would be re-used as common content, while new controls would appear where the style, color and outline groups currently appear. Similar interfaces exist for specifying character and picture symbols as well as all the types of stroke or pattern.

### Discoverable Extension

It would be elegant for this paper if we could use examples of existing discoverable extension methods that control properties on symbol classes in order to best illustrate its usefulness in connection with an extensible symbology. Unfortunately, these haven't been written yet. However, we can gain important insights by drawing an analogy from an extensible data provider interface instead. At the time of submitting this paper, we have mostly begun using the extension methods for adding topology methods to the IFeature. The original intent of extension methods was to allow an easier way to work with sealed enumerable classes when working with System.Linq. The most common use of extension methods is simply to extend an existing class without changing it. However, the benefit that we have found to be the most useful from an extensibility standpoint is that we can make the interfaces that new developers have to implement much leaner. For instance, an IFeature currently specifies a coupling between vector information and attributes. This would be fairly straight forward to implement. However, adding an apparently simple new method called Intersects on this interface would force every individual that only wanted to support a new data format to also implement their own intersection code. Since topology methods are quite advanced and require a sizeable infrastructure to draw on, we chose to support the intersect behavior using extension methods. Any IFeature can now intersect with another IFeature, though it must yield to the definition provided by the extension method.

The danger of extension methods is that they can only be replaced with 'new' functionality and can't ever be overridden. This means that if a feature class implements its own Intersects logic, if an instance of that class is cast as an IFeature interface, then it will call

the Intersects extension method, and not allow the new logic to replace the built in logic.

## Summary and Conclusions

Symbology is a critical part of feature representation, but in both proprietary and open source GIS systems, this fundamental aspect of the representation is seldom extensible. The usual mechanism for managing extensibility tends to be limited to allowing automation of repetitive tasks, working well within the existing data management, analysis and rendering architectures. The extensible architecture used by this version of MapWindow allows symbolizers to be discovered at run-time, opening up a vast new domain for customization and personalization of the open source framework. The user interface design allows a coupling between customizable forms and smart default controls that work reasonably well even if no custom form is provided. The difficulty of working with open ended interfaces programmatically can largely be addressed by the introduction of new extension methods that access or set values. These extension methods can function at many different levels.

Harold A. Dunsford Jr.  
 Department of Geosciences  
 Idaho State University  
 Idaho, USA  
[dunsharo@isu.edu](mailto:dunsharo@isu.edu)  
 Daniel P. Ames P.E.  
 Department of Geosciences  
 Idaho State University  
 Idaho, USA  
[amesdani@isu.edu](mailto:amesdani@isu.edu)

## Bibliography

- [1] D. P. Ames. *MapWinGIS Reference Manual: A function guide for the free MapWindow GIS ActiveX component*. Lulu.com, Morrisville, North Carolina, 2007.
- [2] H. Dunsford. Restructuring of the mapwindow gis project. *PositionIT*, April/May:54-59, 2009.
- [3] H. Dunsford et al. Community code development: A new paradigm for geospatial software in support of the data for environmental modeling(d4em) project. In *AWRA Spring Specialty Conference GIS and Water Resources V*, San Mateo, California, 2008.
- [4] Microsoft. Extension methods (c# programming guide). 2008. URL <http://msdn.microsoft.com/en-us/library/bb383977.aspx>.
- [5] MSDN. Explicit interface implementation: C# programming guide. 2008. URL <http://msdn.microsoft.com/en-us/library/ms173157.aspx>.

This PDF article file is a sub-set from the larger  
OSGeo Journal. For a complete set of articles  
please the Journal web-site at:

<http://osgeo.org/journal>



## Imprint

### Editor in Chief:

Tyler Mitchell - [tmitchell AT osgeo.org](mailto:tmitchell@osgeo.org)

### Assistant Editor:

Landon Blake

### Section Editors & Review Team:

Eli Adam

Daniel Ames

Dr. Franz-Josef Behr

Jason Fournier

Dimitris Kotzinos

Scott Mitchell

Barry Rowlingson

Jorge Sanz

Micha Silver

Dr. Rafal Wawer

Zachary Woolard

### Acknowledgements

Daniel Holt,  $\LaTeX$  magic & layout support

Various reviewers & writers

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this journal, the  $\LaTeX 2_{\epsilon}$  style source has been kindly provided by the GRASS and R News editorial boards.



This work is licensed under the Creative Commons Attribution-No Derivative Works 3.0 License. To view a copy of this licence, visit: <http://creativecommons.org/licenses/by-nd/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.



All articles are copyrighted by the respective authors. Please use the OSGeo Journal url for submitting articles, more details concerning submission instructions can be found on the OSGeo homepage.

Journal online: <http://www.osgeo.org/journal>

OSGeo Homepage: <http://www.osgeo.org>

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8



ISSN 1994-1897



[osgeo.org](http://osgeo.org)