OSGeo Journal Volume 8

February 2011

FOSS4G 2009
Conference Proceedings

OSGeo Community
News & Announcements
Case Studies
Integration Examples

FOSS4G
DENVER 2011
SEPTEMBER 12-16

The Annual International

FREE & OPEN SOURCE
SOFTWARE FOR GEOSPATIAL

Conference Event

2011.FOSS4G.ORG

OSGeo
Your Open Source Compass

Volume 8 Contents

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 sections. 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.[1]

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.[2] 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.[3] 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!

[illegible]

Tyler Mitchell
`tmitchell@osgeo.org`
Editor in chief, OSGeo Journal
Executive Director, OSGeo

[1]The direct URL for article submission is: `https://www.osgeo.org/ojs/index.php/journal/author/submit`
[2]FOSS4G 2011 Denver: `http://2011.foss4g.org`
[3]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[23], the ICA (International Cartographic Association) working group on open source geospatial technologies[24]) 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

[23]OSGeo: Open Source Geospatial Foundation: http://osgeo.org
[24]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 organising 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

A Modular Spatial Modeling Environment for GIS

Brian Marchionni & Daniel Ames, Idaho State University

Abstract

Development of an open source modeling environment for use with spatial-temporal data in a Geographic Information System (GIS) is presented. MapWindow GIS, a free and open source desktop GIS, has been used extensively in watershed modeling and is the underlying engine of the U.S. EPA BASINS system. To date, legacy versions of MapWindow have lacked an integrated modeling environment suitable for linking together geospatial and temporal independent processes at a granular level. Development efforts focused on creating an extensible graphical, open source modeling environment with easy to use programming objects.

This development was made possible due to the new design of the MapWindow GIS 6 project. This new modeling environment allows users and developers to easily create models which can take advantage of spatial and temporal data objects and analytical tools. The design approach involves the extensive use of interfaces, which are essentially skeleton programming tools that detail how an object programmatically interacts with other objects, but not necessarily how it works internally. By using interfaces, the new MapWindow GIS modeler makes it relatively simple to take existing modeling processes, wrap them in an appropriate interface, and execute them as part of a more complex model.

The central underlying design consideration of the newest version of MapWindow GIS was to keep the entire project as modular as possible, this has been extended to encapsulate the development efforts of the modeler as well. The new modeling environment allows developers to automatically generate user interfaces for their processes. Because all tools in the MapWindow modeler must implement the same interface, developers wishing to use a tool directly in their own application need not add the graphical modeler if they do not so desire.

MapWindow GIS 6 and the modeler are entirely developed using the Microsoft .NET Framework which allows it to be run on a variety of operating systems including Windows, Linux or OS X (via the Mono compiler).

Introduction

The goal of the project described here was to create a cutting edge open source modeling environment that worked within the Microsoft .Net framework and contained both a graphical user interface and easy to use programming object. The project was designed to run along side the next generation of the MapWindow project, MapWindow GIS 6. MapWindow GIS 4 is the current version of the project and is under continued development at Idaho State University in the Department of Geosciences. MapWindow GIS 5 was a short-lived prototype project that was never released publicly.

Originally developed at Utah State University, and now maintained primarily at Idaho State University with an international development team, MapWindow GIS is a free and open source software project that is downloaded over 6000 times per month. It has an active community of users and developers on the MapWindow.org web site. The community collaborates on making updates and introducing new features.

The existing project is divided into two components: the MapWindow GIS desktop application, and the ActiveX map control. These two components work together to form the entire project. This modularity allows the ActiveX control to be used in other stand alone applications as well as within the main MapWindow GIS desktop application. (1)

The need to develop a modeling environment arose from other developments in the GIS community. A general need to simplify the task of using spatial and temporal processes had been brought forward by many MapWindow users and by several other communities who are using the MapWindow components in their own projects. Furthermore, the use of modeling in a GIS environment has been suggested by several other researchers including Xie and Brown in their 2007 paper noting, 'simulation in spatial analysis and modeling has been one of the key approaches of many researchers of GeoComputation' (9).

The modeler requirements are closely tied to other developments in the MapWindow 6 project and include:

- all user interfaces need to be as simple to use and as well documented as possible;
- users should need no programming experience to use the software;
- the software must have high portability: software should work on many different systems including MS Windows, Linux and Macintosh OS X;
- the code needs to be highly extensible and reusable;
- the code should be easy to maintain for new developers.

The modeling environment should also be designed such that it can be integrated into other applications with minimal dependency on external libraries, including the MapWindow library itself. Since the design of MapWindow GIS 6 was well underway at the time of the modeling environment's conception, it was decided that components and data types from this new architecture would be used because of the advantages that it afforded, including being memory managed and highly extensible. The data types are not directly link to their sources, which allows many different data formats to be present as a single data type.

MapWindow GIS 6

MapWindow GIS 6 is the next generation of the MapWindow open source project. Early in the planning stages of MapWindow GIS 6 it became apparent that the technology behind the original MapWindow ActiveX map component would not be capable of meeting all of the new project's requirements. Specifically since the original code was written as a Microsoft COM object it could never be cross platform compatible. For this reason it was decided that a complete rewrite of the map component would be required.

The design of the new architecture focused on an extremely modular system using class interfaces. Figure 1 highlights the interface architecture of MapWindow GIS 6. This design allows for any single component to be replaced by another component that uses the same interface. This design stemmed from the successful plug-ins methodology from the original MapWindow GIS 4 that allowed third party developers to extend the functionality of the application by writing their own class which implements the plug-in class interface (2). The improvement presented in MapWindow GIS 6 is that this interface based architecture is extended to every modular component of the architecture and not just to plug-ins.

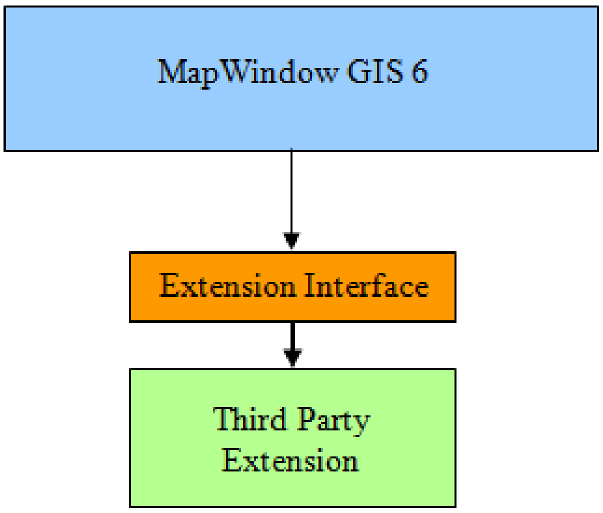


Figure 1: MapWindow GIS 6 Architecture

The MapWindow Modeler Project

Project Requirements

The MapWindow Modeler environment has been developed specifically to meet the requirements of several use cases identified by the United States Environmental Protection Agency (EPA) Data for Environmental Modeling (D4EM) project. Some of the key requirements and constraints of the system are defined as follow:

- the tool should be written in Microsoft .NET so that it can be ported to Windows Mobile and Mono for Linux;
- the available tools and available data types should be extensible;
- the tool should integrate both spatial and temporal components;
- the tool should be easy to use for end users;
- the tool should be compatible with existing versions of MapWindow GIS;
- the tool should be robust and easy for new developers to add to and enhance.

Several existing open source projects were identified to see if they could meet the requirements for a modeling tool for MapWindow. Sextante (7) meets some of the requirements, however it lacks temporal data type support and is written in the Java language and hence would not meet the requirement of being written in Microsoft .NET. No other open source modeling products that were available were written in Microsoft .NET and could handle both spatial and temporal data interaction. The OpenMI system (4) was examined, but it too failed to meet all of the requirements of the system as its scope was well beyond a simple graphical tool for linking spatial and temporal processes, but rather is designed to link larger complex models.

Use Cases

There are three primary use cases for the modeling environment. The first covers the modelers' use while integrated into MapWindow GIS 6. In this mode a standard extension to the MapWindow GIS 6 desktop application will include the modeling environment. These two environments are tightly linked to allow data from the MapWindow GIS 6 map component to be added seamlessly from the modeler, and conversely they allow data from the map to be used in models. The second use case involves integration with the legacy code of MapWindow GIS 4. This is similar to integration with version 6 of MapWindow, but only a specific subset of the data will be made available to the MapWindow GIS 4 application because of format compatibility issues. The final use case covers using the modeler as a stand alone component for use in third party applications. Since all possible uses of the modeler in other applications cannot be considered, the modeler must be as versatile and customizable as possible.

Software Design Technique

Since many different users and developers will be working with the system, the initial design specifications needed to be well defined at the outset. Once this initial design was completed a small group of developers created a set of simple tools to test the general design. It was at this stage that critical modifications were made to the design to address specific problems that developers and users were facing.

The rapid prototyping development technique allowed for feedback from testers and other developers while ensuring a quick time to deployment. Initial development efforts took only six months. Once this critical initial development stage was completed the second phase continued until such time as all parties involved were satisfied with the resulting architecture. Once completed most major design considerations were done and the overall architecture finalized. While changes can still be made at this point they must take into account the existence of other dependant components that need to be integrated and cannot have their functionality impaired. For example, if a new version of an interface is created once this second stage of development is completed, it must ensure that any components using the already existing interface must continue to function seamlessly.

This second stage of development is potentially the most important. Feedback from developers creating tools that will ensure the ease of use of the interface for new developers wishing to create tools or data types for the system.

Software Design

Modeler Design

The MapWindow modeler is composed of two interrelated parts: the ToolManager and the Modeler. The ToolManager lists all of the available tools to the user while also providing access to tools in the Modeler. The Modeler displays, loads, saves, and executes models in a graphical environment.

The Modeler and ToolManager itself are actually .NET form components. Like other programming objects in the .NET environment they have a graphical representation that allows programmers to drag and drop it onto a form without writing any code. This greatly reduces the time needed for programmers to develop an application that uses the modeler. Figure 2 shows the class diagram on the ToolManager and Modeler. Many of the classes are interdependent and used by both components. Figure 3 shows an instance of the ToolManager on the left running within MapWindow 6 displaying the tools it has found, and the Modeler on the right displaying a simple model containing one tool.

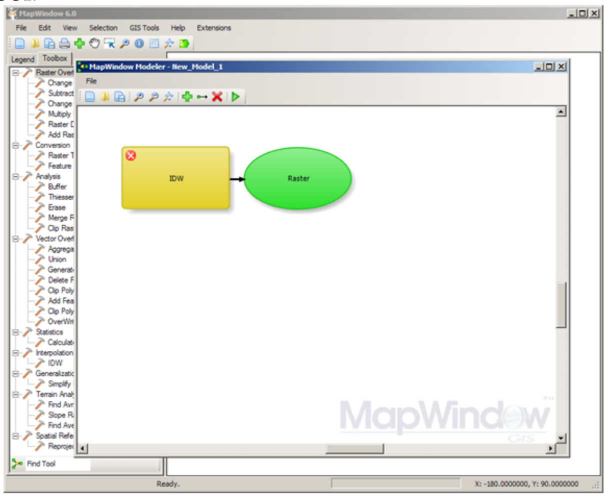

MapWindow 6.0
File Edit View Selection GIS Tools Help Extensions
Legend Toolbox
MapWindow Modeler - New_Model_1
File
IDW
Raster
MapWindow
Find Tool
Ready.
X: -180.0000000, Y: 90.0000000

Figure 3: The ToolManager and Modeler running with MapWindow 6 on Microsoft Windows

Building for Extensibility

Since the use cases for the modeler cover many different applications it was imperative that the modeler be designed such that it can be extended in several different ways, such as:

- tool definitions;
- parameter definitions;
- user interface representation of parameter definitions.

To allow each of these areas to be expanded upon, several programming concepts needed to be employed. These concepts are widely used through the architecture of MapWindow GIS 6 so programmers familiar with this environment can more easily add functionality to the modeler.

To accomplish this, a class interface was defined for tool definitions and parameter definitions called ITool and IParameter, respectively. Using interfaces, blank class templates which programmers can populate with functions (5), allows developers to rapidly develop software which implements the needed operations of software they are interacting with (3). 'A well-recognized method for reducing program complexity involves structuring the model as a set of distinct modules with well-defined interfaces' (6).

Since tools and parameter types can be generated in a variety of different ways, the Modeler never loads ITools or IParameters from disk directly; rather, it relies on a ToolManager to handle loading, and instantiating tools, and parameter types as needed. The ToolManager loads tools by scanning specified folders for assemblies that implement the IToolProvider interface. Once a class that implements this interface is found, it is instantiated and queried for a list of the tools it

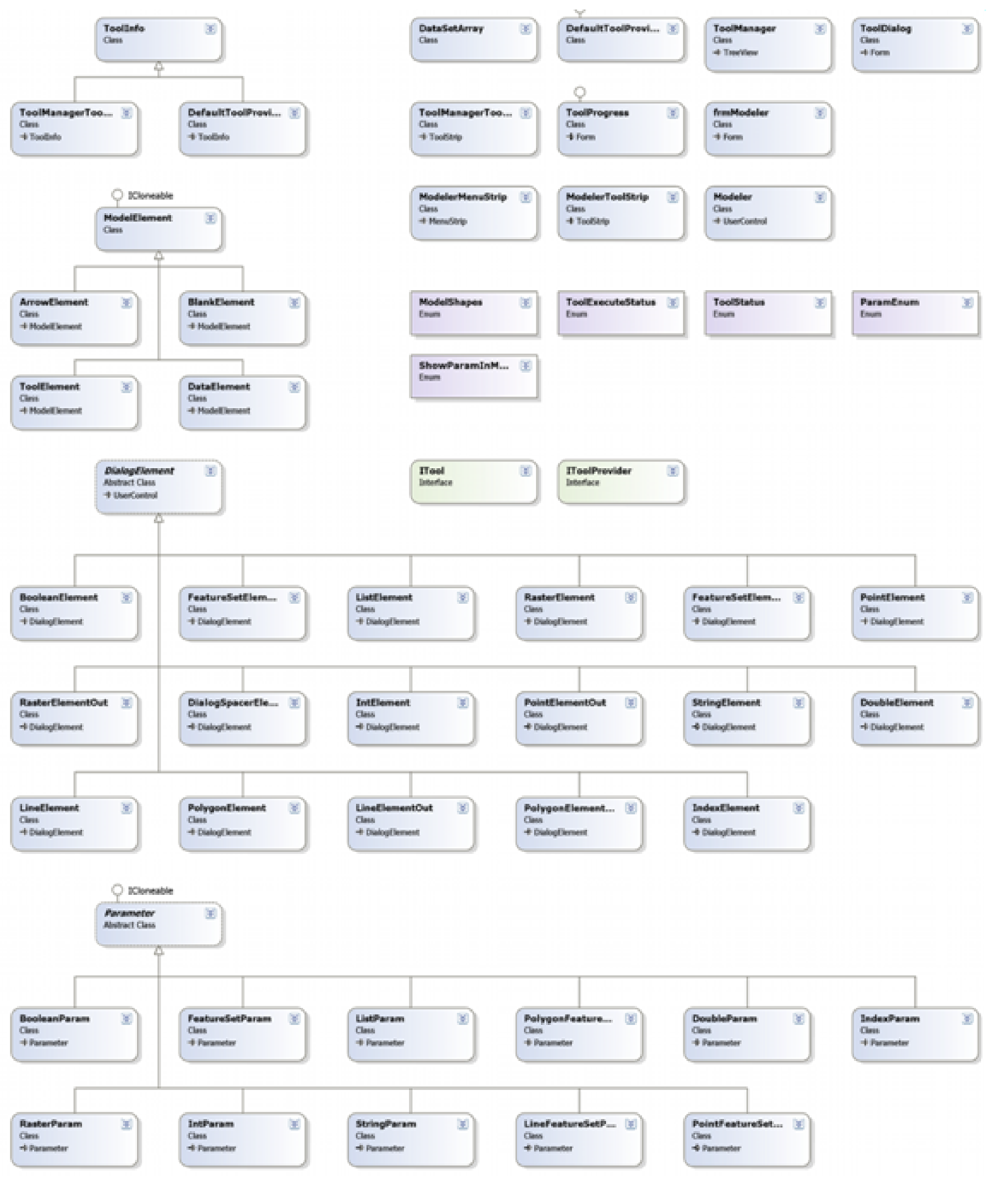


Figure 2: The ToolManager and Modeler class diagram

is capable of providing. This allows tool providers to create tools from a wide variety of sources. A default tool provider is included in the ToolManager. This tool provider scans specified folders for assemblies which implement the ITool interface directly. Loading of parameter definitions and their user interface is also done the same way with the ToolManager looking for assemblies that implement the IParameterProvider interface, and a default provider which scans for IParameter implementing assemblies.

The ITool and IToolProvider Interfaces

The goal of the ITool interface is to remove the burden of creating a user interface and maintaining tool interoperability from the tool developer. Developers designing tools need only implement the ITool interface when designing their tool, and the ToolManager generates the graphical user interface automatically for them when the tool is instantiated. The ITool interface contains several properties which are read by the ToolManager when the tool is first detected and instantiated.

The Name, UniqueName, Category and Version properties on the ITool interface are used by the ToolManager to identify the tool and are require for a tool to be loaded, if any of these fields are missing the tool will not be added to the toolbox. HelpText, HelpImage, HelpURL, Author, Icon and ToolTip are used to populate related parts of the graphical user interface on behalf on the developer. These are optional and ignored if they return null.

The InputParameters and OutputParameters properties return arrays of type IParameter, which are used by the ToolManager and Modeler to execute tools, and populate user interface dialogs with appropriate graphical components. When a tool is instantiated for execution, the initialize method is called first, allowing the tool to populate its InputParameters array with blank parameters. Then, as the user modifies inputs in the array, the ParameterChanged method is called allowing tool developers to create inputs which are dependent on changes to other inputs. Finally when the tool is ready to be run the Execute method is called. The result of this execution is stored in the OutputParameters array and can be transmitted to the next tool by reference or saved to disk as needed. Figure 4 displays the methods and properties of the ITool interface.

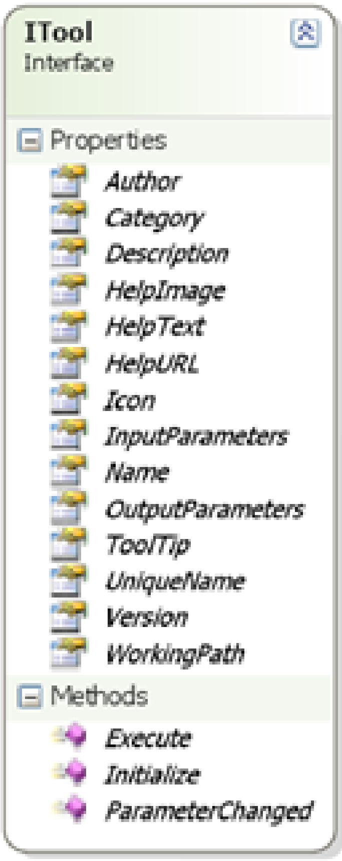
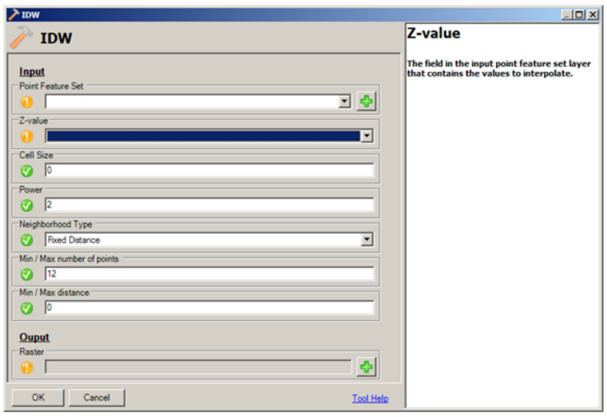


Figure 4: The methods and properties of the ITool interface

Figure 5 illustrates the form that is automatically generated when the Inverse Distance Weighting tool is created. Note the help text on the right is automatically displayed when the user highlights a particular input parameter. Status lights on the left side of the parameter field display the parameters' validity. The ITool interface contains all the information necessary for running and displaying a tool.

The IToolProvider interface allows tools to be generated in a wide variety of ways. While the default ToolProvider searches folders for assemblies that contain ITools, there are many other ways that tools could be generated. For example, a ToolProvider could connect to a Web Processing Service, get a list of available tools, and then generate a corresponding set of ITools which would then be in charge of instantiating for the ToolManager.


IDW

Input

Point Feature Set

Z-value

Cell Size: 0

Power: 2

Neighborhood Type: Fixed Distance

Min / Max number of points: 12

Min / Max distance: 0

Ouput

Raster

OK Cancel Tool Help

Z-value

The field in the input point feature set layer that contains the values to interpolate.

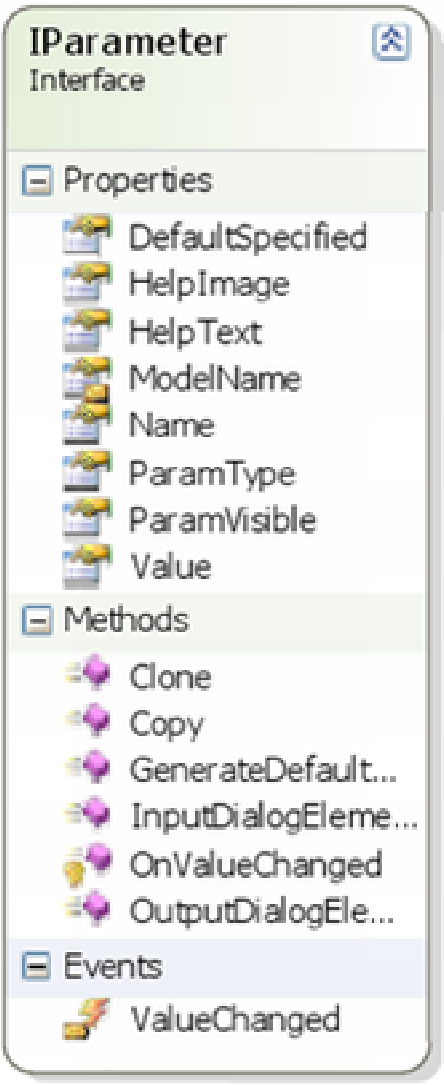
Figure 5: The Inverse Distance Weighting tool dialog running in Window

Tools can also be generated by the Modeler. This is done by saving the model which includes several tools to a XML file which are then recognized by ToolManager as a stand alone Tool. These tools, when called to be executed by the ToolManager will create a new instance of the Modeler, load the saved model and execute it seamlessly as if it were a single tool. This new tool will run as long as each of the Tools used to create the model are available to the ToolManager at execution time.

The IParameter Interfaces

Parameters are the input and output of a tool and need to be defined so that they can have an appropriate visual representation. For example, a numerical parameter should allow for a minimum and maximum value to be specified in order to limit the users' input to a certain range. It should also be capable of specifying a default value and be represented on the tool dialog by a text box that will only accept numerical values. This can be accomplished by creating a parameter object that specifies these constraints and contains a control object that represents how the parameter should be represented in the tool dialog.

The IParameter interface consists of several properties which are used by the ToolManager and Modeler to identify the parameter type. The DefaultSpecified property is used to determine if a tool developer has specified a value to be used as default. The HelpImage and HelpText properties are used to populate the help area on the right hand side of the automatically generated tool dialog. ModelName is used by the modeler to store a unique identifier for a particular instance of a tool. Name is used to identify the parameter input in the tooldialog. The ParamType property returns a string which is used to identify the parameter's type and determine if it is compatible with another tool's input. ParamVisible is used on input parameters to determine if they will be graphically displayed in the Modeler. Finally the Value property contains the actual parameters value.


IParameter
Interface

Properties
- DefaultSpecified
- HelpImage
- HelpText
- ModelName
- Name
- ParamType
- ParamVisible
- Value

Methods
- Clone
- Copy
- GenerateDefault...
- InputDialogEleme...
- OnValueChanged
- OutputDialogEle...

Events
- ValueChanged

Figure 6: The properties, methods and events of the IParameter interface

The clone and copy methods are inherited from the ICloneable class and are used for creating temporary instances of the Parameter when editing values so that if the user cancels without saving his changes they will not affect the underlying objects. GenerateDefaultOutput populates a parameter with a basic value such that the model can be run. InputDialogElement and OutputDialogElement return an instance of the appropriate graphical representation of the parameter for inputs and outputs respectively. OnValueChanged is called when then parameters value have been modified and fires the ValueChanged event, which is then used to notify the IParameter's parent ITool that its value has been modified by calling the ParameterChanged method. Figure 6 displays the IParameter Interface including its properties, methods and events.

Figure 7: The IParameter base element as it appears in the Microsoft Visual Studio designer. The base component is never seen in the modeler

Figure 8: The List Parameter input element as it appears in the Microsoft Visual Studio designer

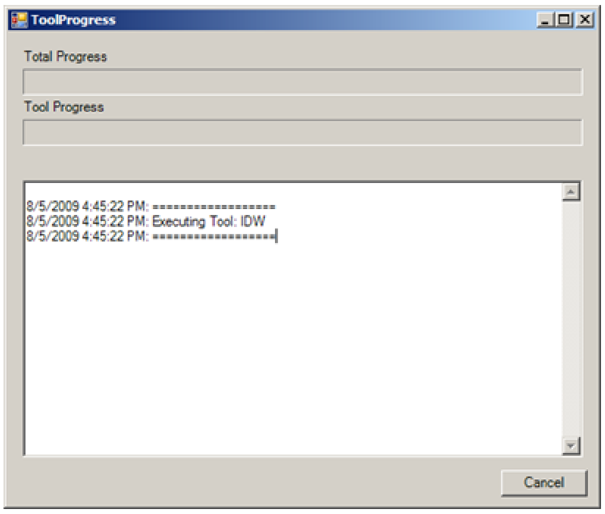
Figure 7 displays the IParameter base graphical user interface, which all parameters must return when the InputDialogElement or OutputDialogElement are called. Figure 8 displays the List Parameter component which implements the IParameter interface. IParameters are responsible for generating two graphical components, one for input and one for output parameter configurations. This ensures that parameters act differently for inputs and outputs.

Tool and Model Execution

There are two different ways that a tool can be executed, either from the ToolManager or from the Modeler. To execute tools from the ToolManager, a user double clicks the tool's name to create a tool dialog populated with the tool's inputs and outputs. Next they populate these fields and then press the ok button. The ToolManager, then calls the Execute method on the tool and displays a progress dialog box.

The modeler executes tools in a similar way. The user drags and drops tools from the ToolManager into the Modeler and links them together by dragging link lines. Internally the Modeler creates association between the relevant input and output parameters. The user can then modify a tool's parameters by double clicking on the tool's graphical representation to access the corresponding tool dialog. Once the model is configured the user clicks the Modeler's execute button which begins the model's execution.

Once a tool is called to be executed, either from the ToolManager or integrated into a model, a background thread is started to carry out the tool's execution. In the modeler, tools that are not ready to be executed because they depend on other tools are queued while tools that are ready to be executed are assigned to a thread and executed. Queued tools are then reviewed as executing tools complete. A separated thread is used to ensure that the tool progress dialog remains responsive to user activity. Messages from the background thread are relayed to the foreground progress dialog thread to allow progress indicators to be updated by the tool. Figure 9 displays the progress indicator form running a tool. In the event of user cancellation, tools are responsible for cleanly exiting.



```
8/5/2009 4:45:22 PM: ===================
8/5/2009 4:45:22 PM: Executing Tool: IDW
8/5/2009 4:45:22 PM: ===================
```

Figure 9: Progress indicator dialog

Modeler Architecture Overview

The Modeler and ToolManager are intentionaly modular, and any single component can be replaced with another, which satisfies the interface requirements of that component. Not all components are necessary for the entire environment to work. If, for example, the Modeler was not included in a project because it was not needed, the ToolManager could be included by itself as a visual component or as a instantiated object invisible to the end user. This high level of interchangability ensures that the components of the modeling environment can be used to meet the widest ranges of developer needs.

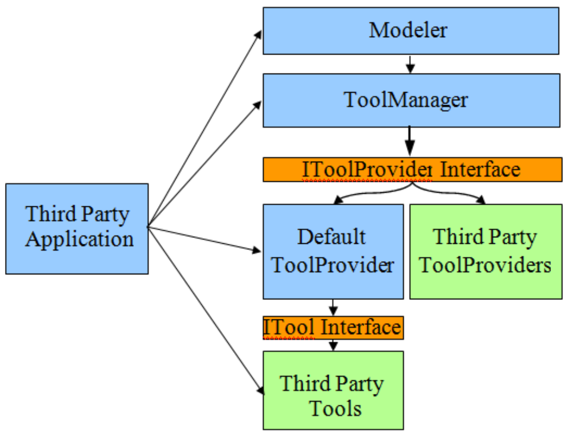


Figure 10: MapWindow Modeler Architecture

Figure 10 displays the overall architechture of the entire MapWindow Modeling project. At the highest level is the Modeler which can be used graphically and programmatically to load, link and execute tools. The Modeler requests instances of tools from the ToolManager which is capable of creating instances of tools and passing them to the Modeler or executing them directly. The ToolManager can load tools using the IToolProvider interface from the Default ToolProvider or from Third Party ToolProviders. The Default ToolProvider can load tools from assemblies which implement the ITool interface directly. Finally third party applications can use any of the MapWindow Modeler components in their own code or they can link directly to Tool or ToolProviders assemblies.

Modeling Environment Comparison and Case Study

The MapWindow Modeler user interface works much like the user interface of the ArcGIS ModelBuilder and gvSIG Sextante Modeler. All three environments present a graphical user interface that allows the user to use the mouse to drag model components from a list into the modeling area where it is represented by a square or circle. In all three environments double clicking on a model element such as a tool or data item, opens a dialog box with options relating to the element. The forms have slightly different appearance but the same functionality. All three environments have help text available to guide the user when configuring a model's elements.

A common task that is often performed when dealing with flood data sets is the delineation of watersheds from raster elevation data. One of the first steps of this process is the generation of raster stream data (8). The initial digital elevation model (DEM) data often contains artifacts referred to as pits; these pits can interfere with the stream delineation process and are usually eliminated by using a pit filling algorithm.

Next the flow direction of the elevation data is calculated. Flow direction calculates the direction in which a drop of water entering a cell would flow. A flow accumulation layer is created next, that calculates the total number of cells which flow into every cell of a raster. Finally the flow accumulation data is reclassified into a Boolean mask where 1 represents cells that have sufficient flow accumulation to be considered a stream and 0 for cells that do not.

This model was created in the ESRI ArcGIS ModelBuilder, the gvSIG Sextante Modeler and in the MapWindow Modeler. Figure 11 illustrates the model as it appears in the ArcGIS ModelBuilder modeling environment (top), in the Sextante Modeler (middle) and in the MapWindow GIS Modeler (bottom). Note that because of differences in the way the processes work internally, they may produce different outputs. Also note that in some cases different tools had to be used because the exact same tools do not exist in all of the environments. For example the Reclassify tool was used in the ArcGIS model and Sextante model while the raster threshold tool was used in the MapWindow model. In some cases a single tool in one environment is capable of producing the same result as two or more tools in another, such as the Flow Accumulation tool in the Sextante tool which produces the same result as the D8 and Flow path tools in the MapWindow model. Also note that the Sextante model does not show the intermediate or output data in its model only the input data and model processes.

Figure 12 shows the final delineated stream data as created by the ArcGIS ModelBuilder and displayed in the ArcMap (top), as created by the Sextante Modeler and displayed in gvSIG (middle) and as created by the MapWindow Modeler and displayed in MapWindow GIS (bottom). Note that the DEM appears differently in MapWindow GIS as its default elevation symbolizer uses a hill-shading technique.

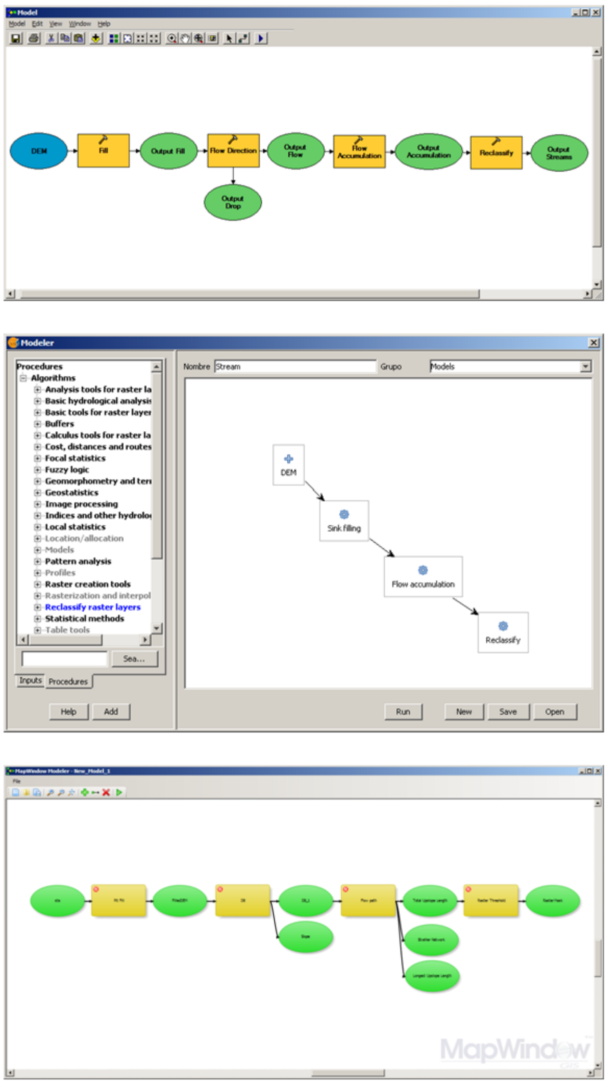


Figure 11: The stream delineation model as it appears in: (top) the ArcGIS ModelBuilder, (middle) the Sextante Modeler, and (bottom) the MapWindow Modeler

Figure 13 shows the results of the three models when overlaid together above the original DEM, note that only small differences exist between the three models, while the general trend of the stream is preserved between them. These slight discrepancies in the various models' results are to be expected given that they each implement slightly different algorithms to achieve their result. The run time of the ArcGIS Modeler was 14 minutes 35 seconds, the run time of the Sextante Modeler was 24 minutes 9 seconds while the run time of the MapWindow Modeler was 22 minutes 33 seconds. The process which took the longest time in all three cases was the pit fill algorithm. It is also responsible for the large discrepancies in times between the three models.

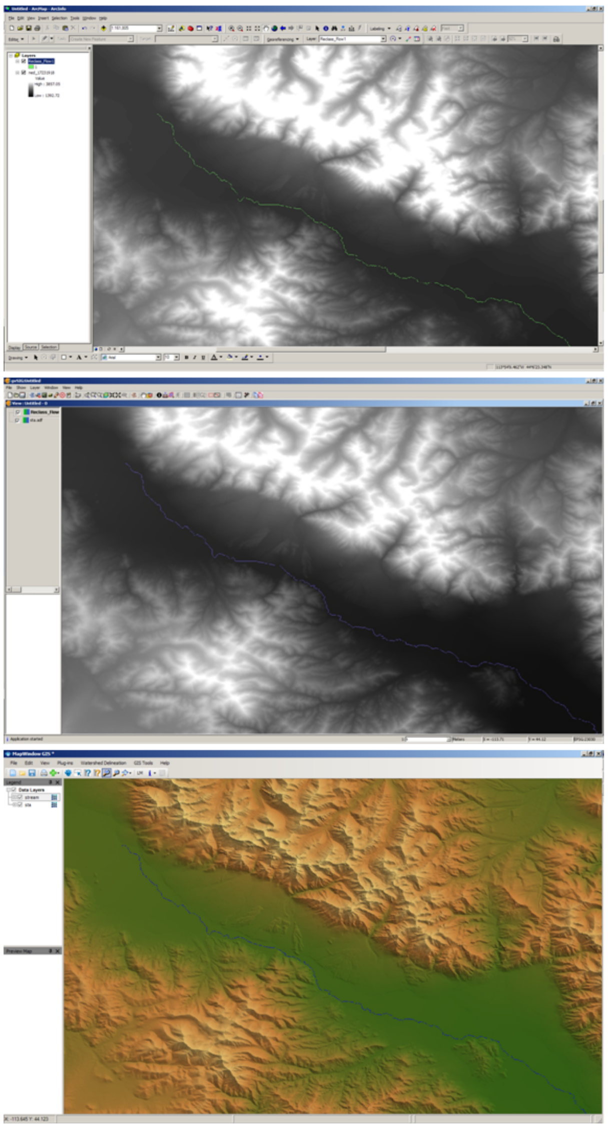


Figure 12: Final stream delineation and original DEM as display in: (top) ArcMAP and produced with ArcGIS ModelBuilder, (middle) gvSIG and produced with Sextante Modeler, (bottom) MapWindow GIS and produced with MapWindow Modeler

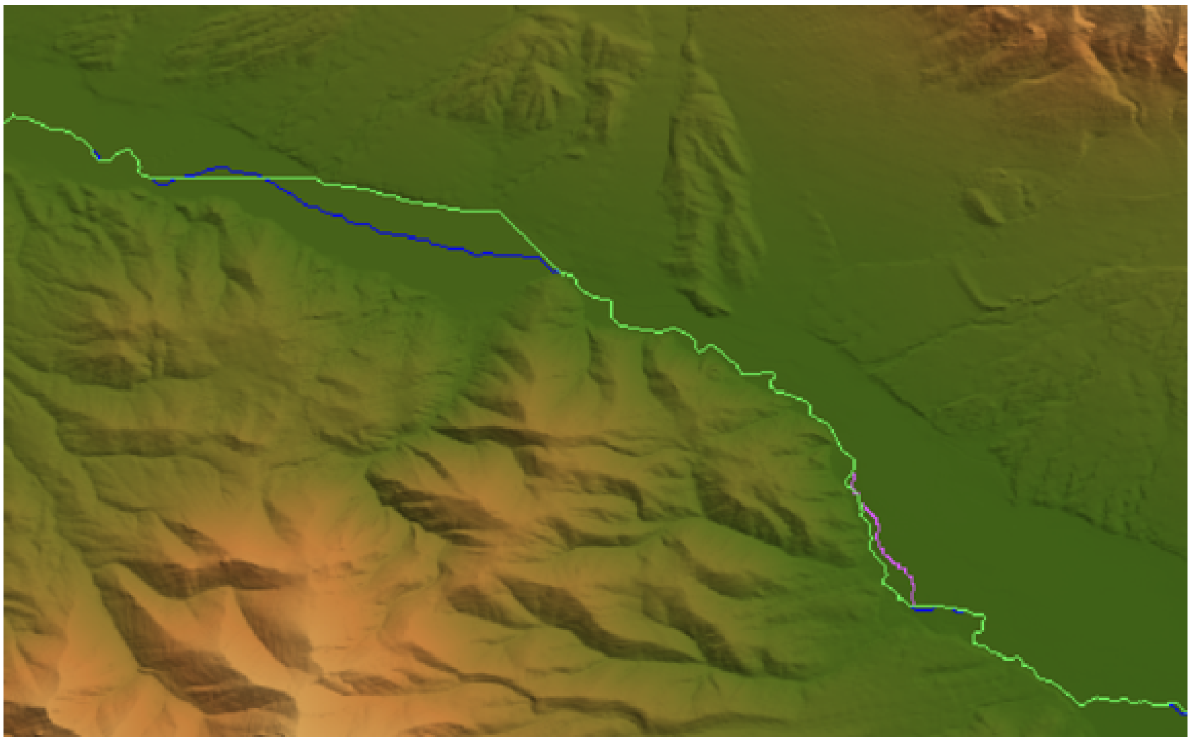


Figure 13: The results of the three models overlaid above the original. Green is the top layer and is the result of the ArcGIS ModelBuilder, Purple the result of the Sextante Modeler and blue the result of the MapWindow Modeler.

Discussion and Conclusions

The MapWindow GIS Modeler is a versatile modeling environment, which can handle many different data types. It executes models in similar time to other GIS modeling environments and faster than other open source ones. Due to its modular and extensible architecture it can use tools of many different designs. The design flexibility not only allows tools to function in a wide variety of different ways, but it allows tools and their associated parameters to be generated from any number of sources. Its ease of use for end users and developers, as well as its integration with MapWindow GIS 6 and MapWindow GIS 4, ensures that the widest range of users will have access to the program. By building on the successful design of previous generations of MapWindow GIS, the MapWindow Modeler benefits from all of the development expertise, keeping the designs that were the most effective while eliminating some of the more constrictive problems. It is one more tool available to both developers looking to create new modeling tools and end users wishing to create models with such tools.

Brian Marchionni & Daniel Ames,
Idaho State University
marcbria@isu.edu
dan.ames@isu.edu

Bibliography

[1] D. Ames. *MapWinGIS Reference Manual: A function guide for the free MapWindow GIS ActiveX map component*. Lulu.com, Morrisville, North Carolina, 2007.

[2] D. Ames, C. Michaelis, and T. Dunsford. Introducing the mapwindow gis project. *The Journal of the Open Source Geospatial Foundation*, 2:13–16, 2007.

[3] S. Greenberg. Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159, 2007.

[4] J. B. Gregersen, P. J. A. Gijsbers, and S. J. P. Westen. Openmi: Open modelling interface. *Journal of Hydroinformatics*, 9(3):175–191, 2007.

[5] L. Liquori and A. Spiwack. Extending feathertrait java with interfaces. *Theoretical Computer Science*, 398(1-3):243–260, 2008.

[6] T. Maxwell. A paris-model approach to modular simulation. *Environmental Modelling & Software*, 14(6):511–517, 1999.

[7] V. Olaya and J. C. Gimenez. *SEXTANTE: a gvSIG-based platform for geographical analysis*. Free and Open Source Software for Geospatial, Victoria, Canada, 2007.

[8] K. L. Verdin and J. P. Verdin. A topological system for delineation and codification of the earth's river basins. *Journal of Hydrology*, 218(1-2):1–12, 1999.

[9] Y. Xie and D. G. Brown. Simulation in spatial analysis and modeling. *Computers, Environment and Urban Systems*, 31(3):229–231, 2007.

Imprint

Editor in Chief:
Tyler Mitchell - `tmitchell AT 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_ε style source has been kindly provided by the GRASS and R News editorial boards.

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

9 771994 189002 00

ISSN 1994-1897

OSGeo

Your Open Source Compass

osgeo.org