

Computational Viewpoint

Captures components, interfaces, interactions and constraints without regard to distribution

MVC in JavaScript

MapBuilder implements the MVC design pattern as a hierarchy of JavaScript Model, View and Controller objects. All objects have properties and methods. Each of the Model, View and Controller base classes provide functionality common across object types. Object properties are documented in the [#ComponentRegister](#) and the methods are documented in the [#JSDocs](#) pages.

Every object must have a unique ID attribute. Every object can be located using the global "config" objects array as in `config.objects.objectId`.

Models

All model objects inherit from the **ModelBase** abstract class. This base class handles all document loading through GET and POST.

The ModelBase class implements the Listener methods for other objects to trigger. The Listener object provides `setParam()` and `getParam()` methods, and notifies Listeners when the parameter changes.

Instances of the abstract base class typically implement get and set methods specific for the model type's schema and namespace. A generic Model object type is provided for when no specialized get or set methods are required.

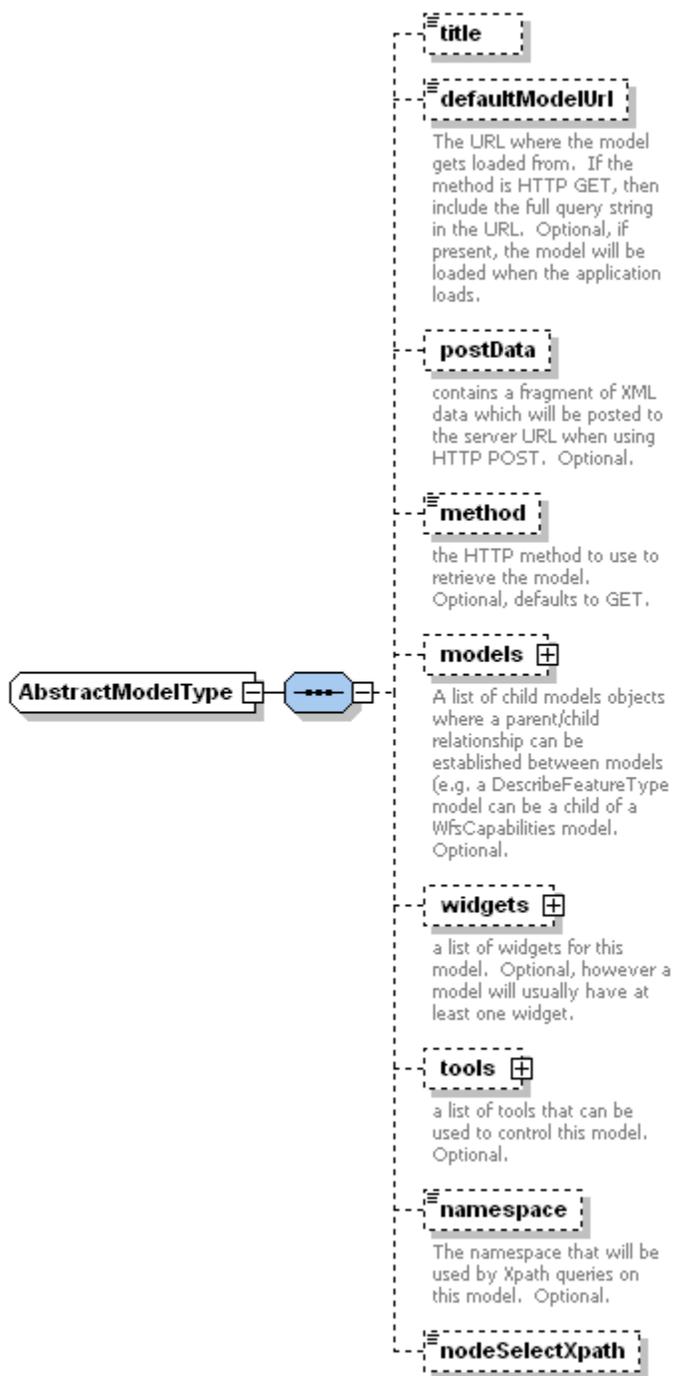
models array

Models also contain lists of child models, widgets and tools. The Model objects in those lists are as described in this section and widget and tool objects are described below. This means that a Model may have child models where it makes sense to do so, e.g. a WFS capabilities model may have child GetFeature, DescribeFeatureType and Transaction models.

The MapBuilder **Config** object is itself a Model object where the document is the config file. The Config model also implements some special bootstrapping methods for object initialization.

Widgets

All widget classes inherit from the abstract **WidgetBase** class. This class handles common widget property handling and selecting the HTML node in the DOM but no `paint()` method. This class must be further derived by a widget class that implement a `paint()` method that handles the actual rendering of the view, which may vary widely (see Graphics Rendering).



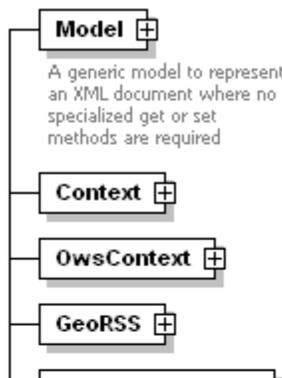
The **WidgetBaseXSL** widgets implement a paint method where the widget output is the result of an XSL transformation on the model document. By default, the XSL filename corresponds to the object type name with an extension of .xsl by default. For example, the <Legend> XSL file to style the Legend object is lib/widget/Legend.xsl by default. This default filename can be over-riden by setting a <stylesheet> property on the widget object.

MapContainer widgets share a map container where 2D rendering of geo data is layered. The container provides an Extent tool that translates screen pixel/line coords into map projected X and Y.

Button widgets are a specialized widget type where the output acts as buttons.

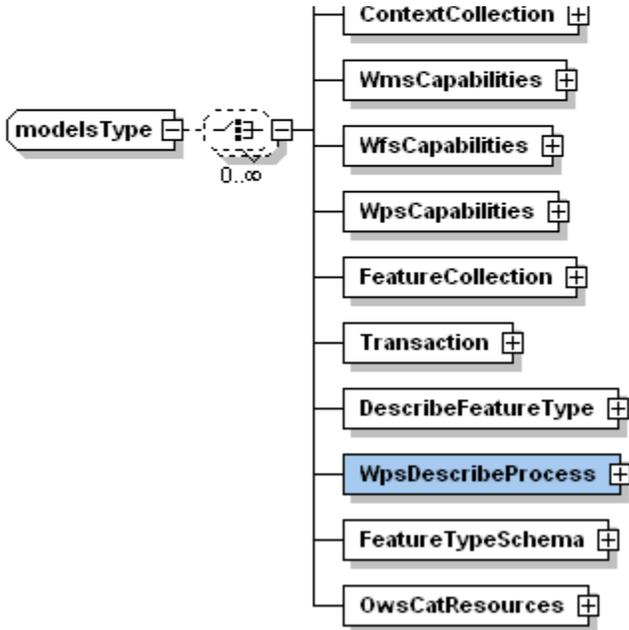
Tools

Most tool classes inherit from the **ToolBase** class. Tools tend to be much more varied in their purpose so there is less commonality among them. This class must be further derived by tool classes that implement the controller methods required.



Some of the tools available are listed here. All tools "control" the model in some way, either by sending events to the model for listeners to pick up on or manipulating the XML document of the model.

An exception for tools is the Extent tool which handles the affine transformation from screen pixel and line coordinates to map projected XY values. This tool



doesn't get included in the config file because it is always required for models that have a MapContainer widget.

Listeners

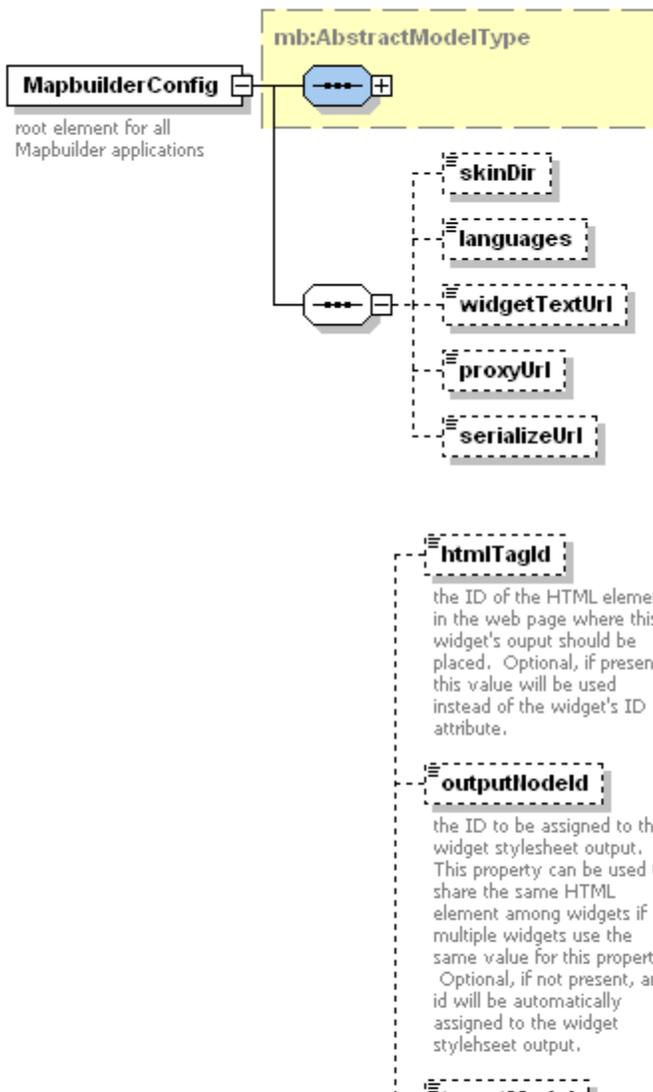
Models inherit from the Listener class. (add graphic)

This is why objRef references are passed in listener functions.

JSDocs

JavaDoc style description of MapBuilder object methods <http://demo.communitymapbuilder.org/mapbuilder/docs/jsdoc/index.html>. This documentation is automatically derived from structured comments in the code which is why it is very important that the code is properly documented.

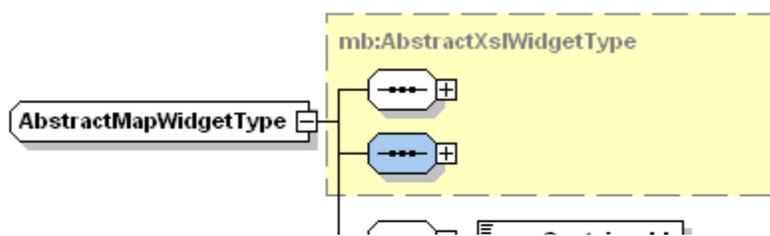
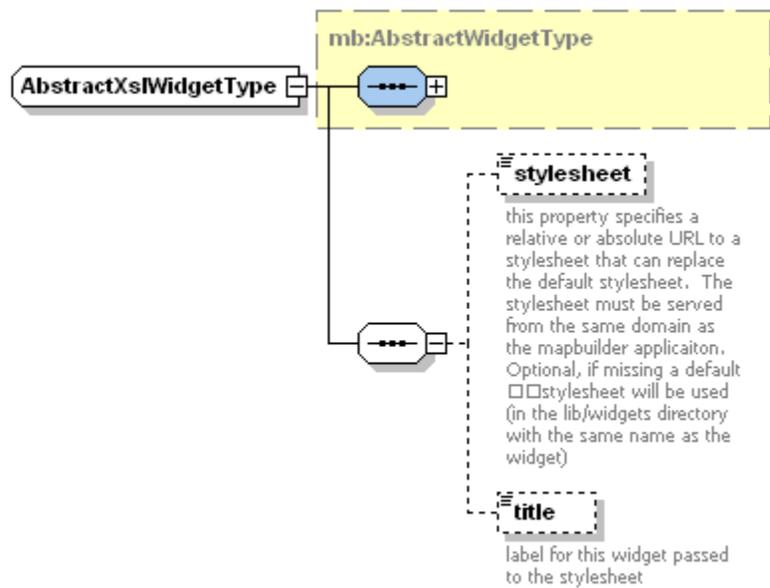
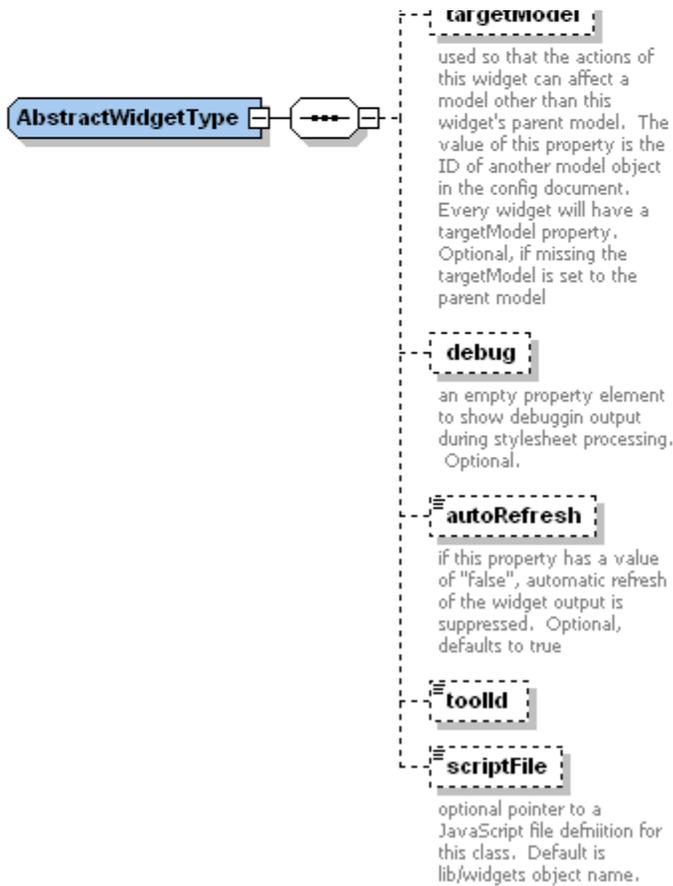
(add outline of JSDoc comment structure)



Component Register

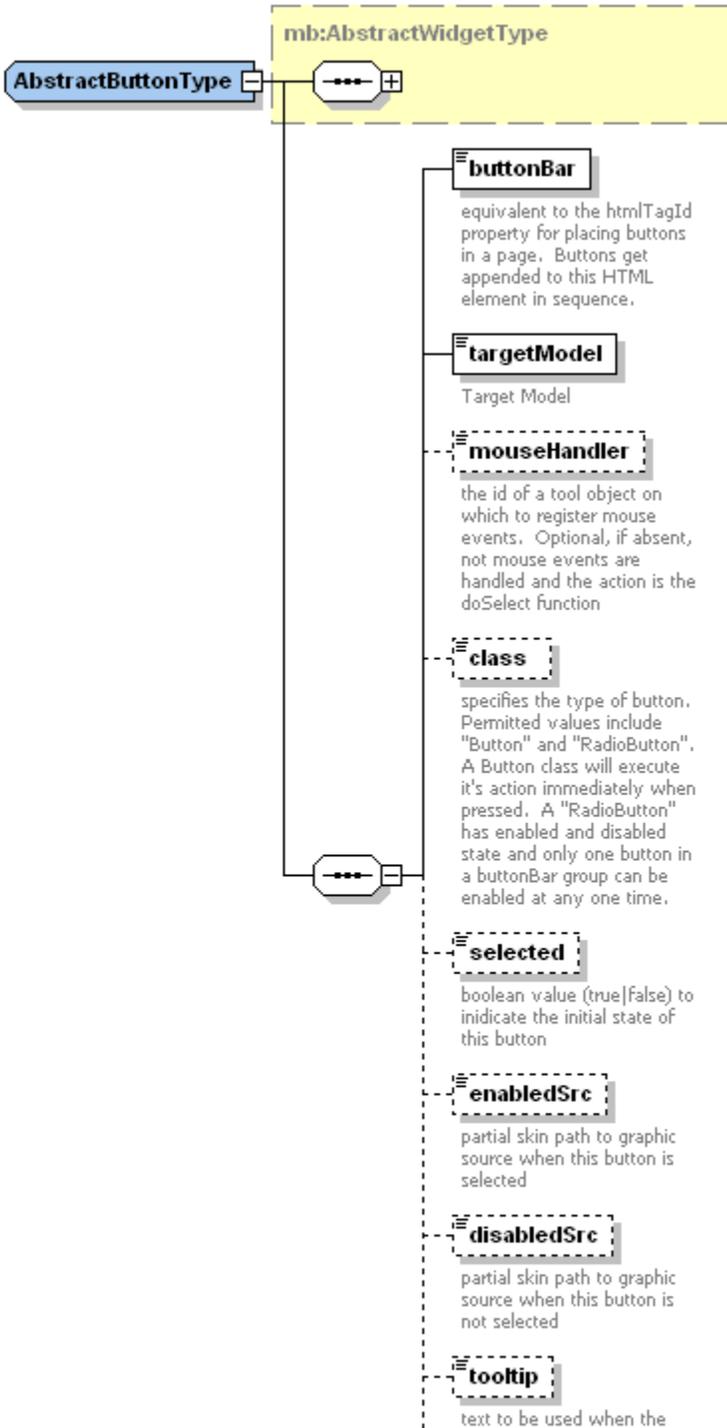
All objects are (or should!) be documented in the configuration file schema which can be found in the source tree at `mapbuilder/lib/schemas/config.xsd`. This schema is used for the [Component Register](#) and describes all object properties.

Configuration documents should be able to validate against this schema, and all official demos in v1.0 do validate. However since custom XSL stylesheet parameters can be set in the configuration file and not necessarily be included in the schema, custom configuration files may not validate. The order of configuration properties also impacts validation.



mapContainerId

Used to assign an ID to a container DIV element for 2 dimensional DIV map widgets. Multiple widgets may share this container for rendering their output. The first model encountered in the config file using this ID becomes the "source" model for this widget so that extent, AOI and other map related data can be shared by multiple models. Required.



mouse hovers over this button.

action

cursor

AbstractToolType

targetModel

the id of a model object which this tool acts on. Optional, if absent the action happens on the parent model of this tool.

enabled

a flag to indicate if this tool should initially be enabled. Optional, accepts values of true and false. Tools are enabled by default.

AoiMouseHandler

AoiBox

DragPanHandler

MouseClickedHandler

ZoomToAoi

toolsType

1..∞

WebServiceRequest

mb:WebServiceRequestType

targetModel

the id of a model object which this tool acts on. Optional, if absent the action happens on the parent model of this tool.

enabled

a flag to indicate if this tool should initially be enabled. Optional, accepts values of true and false. Tools are enabled by default.

requestName

requestFilter

maxFeatures

Caps2Context

EditContext

MovieLoop

