Guide to the Gorm application

Version 1.1.0
(for use with 'gnustep-gui' version 0.11.0)
(and with 'gnustep-base' version 1.10.0)

Gregory John Casamento <greg_casamento@yahoo.com>
Richard Frith-Macdonald <richard@brainstorm.co.uk>

Note: The Gorm application is in alpha release. You will be performing a valuable service if you report any bugs you encounter.

Table of Contents

Copying

See the file ‘COPYING’.

Contributors to Gorm

- Gregory John Casamento <greg_casamento@yahoo.com> Is the current maintaner of Gorm. Has implemented lots of new features and rewritten large portions of the existing code.
- Richard Frith-Macdonald <richard@brainstorm.co.uk> wrote the original version of Gorm as part of the GNUstep project.
- Pierre-Yves Rivaille <gnustep@rivaille.net> Is also a major contributor to the Gorm application.

1 Installing Gorm

1.0.1 Required software

You need to have the GNUstep core libraries installed in order to compile and use Gorm. The core packages are, at a minimum:

- gnustep-make
- gnustep-base
- gnustep-gui
- gnustep-back

See http://www.gnustep.org/ for further information.

1.0.2 Build and Install

Steps to build:

- make && make install

Please note that GormLib must be installed for Gorm.app to run.

1.0.3 Trouble

Give us feedback! Tell us what you like; tell us what you think could be better. Send bug reports and patches to bug-gnustep@gnu.org.

2 News

2.1 Noteworthy changes in version '1.1.0'

- Changed Gorm architecture to use NSDocument classes.
- Abstracted model loading mechanism. This was done by implementing a set of "Loader" and "Builder" classes which handle filling in the data structures in Gorm and exporting them to external formats.
- Implemented GormNibWrapperLoader and GormNibWrapperBuilder for reading and writing Cocoa NIB files.
- Implemented GormGormWrapperLoader and GormGormWrapperBuilder for reading and writing GNUstep Gorm files
- Implemented GormGModelWrapperLoader for reading GNUstep gmodel files.
- Updated icon
- A number of bugs have been addressed in this release.

2.2 Noteworthy changes in version '1.0.8'

This is a bugfix release.

- Correction for bug#16587.
- Correction for handling non-string identifiers in tableviews.

2.3 Noteworthy changes in version '1.0.6'

This is a bugfix release.

- Entirely new icon set, for palettes, gorm, gmodel, nib and the application.
- Replaced some of the images for the inspectors.
- Corrected the following bugs since the last release: #16049, #16050, #15988, #16049, #15989, #15987, #15817, #15780, #15642, #15556.
- Changed formatting in some of the inspectors so that they are easier to navigate.

2.4 Noteworthy changes in version '1.0.4'

This is a bugfix release.

- Corrected some bug#15236 with window style mask settings.
- Corrected bug#15236, which caused window fields in the inspector not to update when the field was being edited and a new window is selected.
- Corrected bug #15178.
- Corrected problem with standalone views

2.5 Noteworthy changes in version '1.0.2'

This is a bugfix release.

- Fixed some bugs with table column selection.
- Corrected a minor problem in the custom class inspector.

2.6 Noteworthy changes in version '1.0.0'

PLEASE NOTE: This version of Gorm requires base 1.11.1 and gui 0.10.1 to be installed (gnustep-startup-0.13.0).

- All inspectors are now modeled in .gorm files.
- Added autosizing to form attributes inspector.
- Utilize and maintain parent/child data structure more pervasively
- Reorganized code in palettes for cleaner implementation.
- Removed code to check for user bundles, since bugs in Camaelon which prompted those changes were fixed long ago.
- Added documentation to GormCore

2.7 Noteworthy changes in version '0.11.0'

- Improved implementation of canSubstituteForClass: the default implementation of this method tests the classes to see if initWithCoder: or encodeWithCoder: is implemented on a subclass to determine automatically if that class has the same encoding signature as the original class, if it does, it can be substituted.
- Improved handling of classes which use cell classes in the custom class inspector. The inspector now autmatically replaces the cell class with the appropriate one when the user selects a given subclass.
- Browser based class editor in document panel. This interface is more like the one on OSX. The user now has a choice in preferences to determine which view they would like to use.
- Translation tools. The Document->Translate menu allows the user to export string and import strings in the strings format, so that someone can easily translate just the strings in the file and doesn't need to directly edit anything in Gorm. The strings file can then be loaded back into Gorm and all of the relevant strings are updated.
- Alignment tools. In the new Layout menu there are options to align views, center views, bring views to front or push them to the back of the view layers.
- Implementation of IBViewResourceDraggingDelegate. This allows updating of the pull down in the inspectors panel dynamically. It requires the developer of a palette to implement some code to enable this, as on OSX.
- Lots of bugfixes and usability changes are also included in this release.

2.8 Noteworthy changes in version '0.9.10'

- Gorm now has a full implementation of canSubstituteForClass: which is used to determine if a class can be substituted in the custom class inspector. This allows classes added in palettes to say whether or not they can be used as a subsitute for a kit class.
- Better separation of Gorm into libraries. As well as the ability to compile on windows with a simple: "make install"
- Implementation of IBResourceManager class. This class is used by palettes to register drag types to be considered by the top level editors in the document window: object, sound, image, class.

- Gorm now is able to switch views in the document window when you drag a file into it. If it's an image it will switch to the image view, if it's a sound, the sound view, an object the object view etc or if it's a class (a .h file) it will switch to the classes view.
- Drag and drop parsing of header files (if you hadn't gathered from the previous item).
- Better support for standalone views. while the user cannot instantiate from the classes view (there were too many problems with this approach). They can now drag any view from the palette into the objects view and have it work.
- A myriad of bug fixes.

2.9 Noteworthy changes in version '0.9.2'

NOTE: This is mainly a bugfix release.

- Some improvements to the procedure for removing connections.
- Corrected various issues with header parsing.
- Now closes windows which were opened during interface testing such as font panels, info panels, etc.
- Minor corrections to background color for a number of inspectors.
- Improvements to gmodel importation.
- Better detection of when the user is utilizing a user bundle. Gorm will now warn the user with a panel.
- Various improvements in documentation

2.10 Noteworthy changes in version '0.9.0'

- Images/Sounds can now be dragged into a matrix cell.
- Fully implemented date and number formatter inspectors (these classes still need work in GUI).
- Added warning panel if the user attempts to edit a .gorm file created with a newer version of Gorm
- Modified data.classes format so that only those actions specifically added to FirstResponder are listed.
- Greatly improved gmodel importation. (experimental)
- It's now possible to add methods to classes which are not custom. This allows the user to add actions which may have been added to those classes by categories.
- Completely new header parser implemented.
- Improved cut/paste. It's now possible to use cut/paste from almost anywhere. The class editor now fully supports it.
- Improved implementation of some of the InterfaceBuilder framework classes.
- Object editor will now remove all instances of a class that has been deleted from the class editor.
- The class inspector and the classes view will now apply stricter rules to names of actions and outlets to ensure that they are properly entered.
- All inspectors work perfectly with customized colors.
- Fixed a number of bugs.

2.11 Noteworthy changes in version '0.8.0'

PLEASE NOTE: It is important for this release that you upgrade to Gorm 0.8.0 when using Gorm with the new GNUstep libraries (base-1.10.0 and gui-0.9.4). This version of Gorm contains some features which are reliant on changes made in those versions of the libraries. It is stated in Gorm's documentation (the Gorm.texi file) that this is required, but I felt it important enough to also mention it here so that it is known beyond a reasonable doubt.

- New gorm file version.
- Full custom palette support
- Palette preferences panel to allow the user to configure palettes to load
- Experimental: Standalone views. This feature is to allow the use of a view without the need of a containing window. This allows developers to treat these views as they would any other top level object in the .gorm file. This is experimental functionality.
- Improved NSTableColumn inspector. The new inspector allows the user to change the data cell used for a given column. This allows the user to select from a list of cell subclasses and set the appropriate custom or non-custom one they want to appear in that column of the table.
- Improved layout of some of the inspectors.
- Removed old class parser. The parser was somewhat buggy and was actually causing some issues. A new parser will be available in the next version of Gorm. For now users will need to use the class inspector or the outline view to enter classes into Gorm.
- Experimental: "File" section. This is essentially a per-file preference which allows the user to control which version of GNUstep a given file will be compatible with. It also lists the potential compatibility issues with the selected version.
- Improved controls palette. New items for some of the standard font replace the old "Title" widget which was a System-14 font. The new widgets use a selection of the standard System font to allow the user to easily build a gui using these and reducing the amount of time the user needs to spend fiddling with the font panel.

2.12 Noteworthy changes in version '0.7.7'

- Important bugfixes in editor classes.
- Rearranged some of the editor classes to be in the palettes which contain the classes they are responsible for editing (GormButtonEditor & GormTabViewEditor).
- Image and Sound editors will now display system default images or sounds if they are available.
- Document window now uses an NSToolbar (experimental).
- Improved the layout of some of the inspectors.
- Corrected some minor issues in the inspectors
- Added code to allow NSTableView and NSOutlineView to show some data during testing
- Gorm will now show an alert panel when a model fails to load or test properly.

2.13 Noteworthy changes in version '0.7.6'

This release is mainly a bugfix release for 0.7.5.

- Improved .gmodel support
- Corrections to previous repair feature.
- Important bugfixes for Menu editing.
- Important bugfixes for class inspector.

2.14 Noteworthy changes in version '0.7.5'

- The 'reparent' feature in the class inspector. This allows the user to change the class hierarchy from within Gorm.
- Some important bugfixes
- a property 'GormRepairFileOnLoad' (untested) which should repaire old .gorm files... It is HIGHLY recommended that Gorm not be run with this on constantly and that you back up any files which you want to repair before opening them with this option turned on.
- A shelf inspector in prefs that lets you expand the size of the names in the object view..
- Support for NSFontManager
- A way to restore a complete NSMenu if it's deleted (a new palette entry for NSMenu, not just an item)

2.15 Noteworthy changes in version '0.6.0'

- Several major bugs corrected.
- Clarified some of the inspectors
- Menu items are now properly enabled/disabled when appropriate
- More descriptive title displayed when a class is being edited.

2.16 Noteworthy changes in version '0.5.0'

- Enabled defer in NSWindow inspector.
- Added code to the connection inspector to prevent erroneous connections.
- Added support for upgrading of old .gorm files using the older template mechanism
- Grouping with an NSSplitView now operates using the relative positions of the views in the window.
- Custom Class inspector now shows all subclasses, not just direct custom subclasses.
- Bug fixes, eliminated memory leak, code cleanup, etc.

2.17 Noteworthy changes in version '0.4.0'

- New Menu and Menu Item inspectors.
- User can now specify the Services and Windows menus in the menu inspector.
- User can specify a non-custom subclass as well as a custom one to replace the class when the .gorm is unarchived. This can be used to turn a NSTextField into NSSecure-TextField and etc.

- New set name panel.
- New switch control on the font panel to allow the user to specify if a font is encoded with its default size or not.
- Added NSStepper and NSStepperCell to the class list to allow creation of custom subclasses.
- Windows and Services menus now function correctly.

2.18 Noteworthy changes in version '0.3.1'

- New custom class system.
- Images now persist correctly when added to a button or view.
- Fixed DND
- Various bugfixes

2.19 Noteworthy changes in version '0.3.0'

- Preferences added.
- User can now enable and disable guidlines for easier editing.
- Refactored code into GormLib which is a clone of the InterfaceBuilder framework. This facilitates creating palettes and inspectors outside of Gorm.
- Added class inspector for easier editing of classes. This gives the user the option to use either the outline view or the inspector to edit new classes.
- Added inspectors for the following: NSScrollView, NSProgressIndicator, NSColorWell, GormImageInspector (for images added to .gorm files).
- Improved look of NSTabView inspector.
- Removed all warnings from the code.
- various bug fixes.

2.20 Noteworthy changes in version '0.2.5'.

Many fixes and improvements to make the app work better.

- Better parsing of headers
- Interface code redone as gorm files.
- Re-add multiple selection via mouse drag.

2.21 Noteworthy changes in version '0.2.0' snapshot.

Gobs of improvements, mostly due to the hard work of Gregory John Casamento and Pierre-Yves Rivaille. Thanks guys!

- Custom class support/translations implemented.
- Added NSScrollView, NSPopupButton, NSOutlineView, NSTableView editing.
- Improved test mode support.
- Improved drag n' drop support on many items.
- Intelligent placement hints.

- Read gmodel files.
- More inspectors.
- Sound and Image support.
- gorm files were changed to directory wrappers for more flexibility.

2.22 Noteworthy changes in version '0.1.0'

- load/parses class files for entry into class list.
- Pallete/inspectors for date and number formatters
- Pallete/Inspectors for browsers and tableViews
- NSStepper, NSForm, NSPopupButton pallete item and inspector
- Most inspectors greatly improved and fleshed out.
- Custom views added.
- Ability to edit cells in a matrix.
- Ability to change the font of some objects.

2.23 Noteworthy changes in version '0.0.3'

- Create stub .m and .h files from new classes
- Works better with ProjectCenter.
- Handle Ctrl-Drag and Alt-Drag of objects - automatic conversion to matrices and/or increase decrease rows and cols.
- Edit NSForms titles in place.
- Edit NSBoxes and add subviews.
- Support for custom objects.

2.24 Noteworthy changes in version '0.0.2'

- Add popup and pulldown menu controls
- Menu support
- More inspectors
- Some support for connections
- Much more fleshed out - too numerous to mention.

2.25 Noteworthy changes in version '0.0.1'

- 8th December 1999
  - Save/Load 'nib' documents (binary archived data)

    This works so far as it can be tested - but that's just archives containing windows or panels so far.
  - Load palettes

    Loading of palettes works. You can load palettes from the 'Tools' menu. Gorm automatically loads all the palettes from its Resources directory.

    - Basic framework

      So far, the app provides a basic framework that needs fleshing out.

        - It has a palettes manager object that allows you to select a palette and drag items from the palette into your document.
        - It has a special per-document editor object, which keeps track of a matrix of icons representing the top-level objects in the document.
        - It has an inspector manager class, which updates the inspector panel when the selected object is changed by an editor.
        - It has special inspectors for handling an empty selection or a multiple selection.
    - Palettes

      Four palettes (three of which are empty at present) are built and installed in the apps Resources directory.

      The Window palette is more fully fleshed out than the other palettes. It permits windows and panels to be created in Gorm. If provides the start of a window attributes inspector.
- 18 December 1999
    - You can drag views from a palette into a window or panel.
    - You can select views in a window by clicking on them, shift-clicking (for multiple selection), or click-drag on the window background to select views in a box.
    - You can delete/cut/copy/paste views betwen windows.
    - You can move views in a window by clicking on them and dragging.
    - You can resize views by clicking on their knobs and dragging.
    - You can control-drag to mark source and destination views for a connection.
    - Next task - inspectors.

      The connection inspector needs to be implemented to complete the process of establishing connections. The size inspector needs to be implemented to set auto-sizing parameters for a view.

      Once these are done, the object editor needs to be made to support connections so that we can connect between objects other than views, then we need to write a menu editor.
- 22 December 1999
    - Connections inspector is now working - but it needs some effort to tidy it up.
    - Class info (outlets and actions) is specified in 'ClassInformation.plist' and needs to be present so that the app knows what outlets/actions an object has (and therefore what connections can be made).
    - The view size inspector is working - allowing you to set the size of the subviews within a window.
    - The attributes inspector for 'FilesOwner' is working, so you can define the class of the files owner (it defaults to NSApplication).
    - There is a crude panel for setting the name of the selected object.
    - I've created a couple of new images and got rid of the two NeXT images that were lurking in there.

    - There is a Testing directory, with a GormTest application that lets you load a nib for testing - it assumes that the nib will set its FilesOwners delegate to point to a window, and makes that window the key window ...

- 23 December 1999

  Last work before christmas ...

  Various bits of tidying up plus -

  Added an evil hack of a generic attributes inspector ... This looks through all the methods of the selected object to find those taking a single argument and beginning with 'set'. It makes all these setting methods (whose argument is a simple scalar type or an object) available for you to invoke from the inspector panel.

  This makes it possible to set pretty much any attribute of any object, but you do need to have the GNUstep header files to hand, so you can tell what numeric values to enter to achieve a desired result.

2.25.1 To Do

- Debug and stabilize existing code.

3 Overview

Gorm is an application for creating the user interface (and possibly entire applications) for a GNUstep application. Initially a close clone of the old NeXTstep 3.3 Interface Builder application, I expect that Gorm will mutate beyond the capabilities of that app.

GNUstep is an object-oriented programming framework and a collection of tools developed for or using the GNUstep libraries.

You can find out more about GNUstep at http://www.gnustep.org

The basic idea behind Gorm is simple - it provides a graphical user interface with which to connect together objects from the GNUstep libraries (as well as custom-written objects) and set their attributes in an easy to use manner.

The collection of objects is then saved as a document which can either be re-loaded into Gorm for further editing, or can be loaded into a running GNUstep application in order to provide that application with a user interface or some subsystem.

3.1 What You Must Know To Understand This Manual

This manual assumes a working knowledge of Objective-C and C. These are necessary prerequisites to understanding some of the technical details and examples given here.

3.1.1 Major features

- Drag-and-drop creation of GUI elements from palettes.
- Run-time loading of additional palettes that may be written using an API very similar to that of Apple/NeXTs interface Builder palette API.
- Direct on-screen manipulation of GUI elements
- Manipulation and examination of objects via inspectors.
- Drag-and-drop creation of connections between objects.
- Interactive test mode for interfaces/object-networks under development.
- Saving data in a format loadable by GNUstep applications.

3.2 About this Manual

This manual is ment to cover basic operation of the Gorm application. It is not meant to be a complete tutorial on GNUstep programming.

4 Usage

Here is a description of the menu structure and what each menu does -

- Info
  The '`Info`' menu item produces a submenu ...
  - Info Panel
    A panel giving very limited information about Gorm
  - Preferences
    A panel allowing you to set preferences about how Gorm operates
  - Help (not implemented)
    A panel providing general help on using Gorm
- Document
  The '`Document`' menu item produces a submenu ...
  - Open
    This produces an open panel that lets you open a Gorm document. You use this if you want to use Gorm to edit an exisiting document.
  - New Application
    This creates a new application document within Gorm, you may then use the Palettes panel to drag new objects into the document.
  - New Module
    Contains a submenu, which also contains:
    - New Empty
      produces an empty document with only NSFirst and NSOwner.
    - New Inspector
      produces a document with NSOwner, NSFirst and a window which is the correct size for an Inspector.
    - New Palette
      produces a document which is like the one by '`New Inspector`', but it's window is the right size for a Palette.
  - Save
    This saves the current document
  - Save As
    This saves the current document to a new file and changes the document name to match the new name on disk.
  - Save All
    This saves all documents currently being edited by Gorm.
  - Revert To Saved
    This removes all changes made to the document sunce the last save, or since the document was opened.
  - Test Interface
    This provides interactive testing of the active document. To end testing, you need to select the '`quit`' menu item.
  - Translate
    Contains a submenu, which also contains:

        - Load Strings
          Load a string file. This file contains the strings to translate.
        - Export Strings
          Export a strings file. TODO
    - Miniaturize
      This miniaturises the active document (or whatever panel is currently key).
    - Close
      This closes the currenly active document.
    - Debug
      Prints some useful internal information.
    - Load Sound
      Loads a sound into the .gorm file.
    - Image
      Loads an image into the .gorm file.
- Edit
  In addition to the usual Cut, Copy, Paste, Delete Select All, this menu also contains:
- Set Name
  This allows the user to set a name for a given object in the Objects view in the main document window.
    - Group
      Which produces a submenu
        - In Splitview
          Groups views into an NSSplitView. Gorm does this based on the relative positions of the views being grouped. It determines the orientation and the order of th views and then groups them either vertically or horizontally in the order they appear on the screen.
        - In Box
          Simply groups all of the views into one NSBox.
        - In ScrollView
          Simply groups all of the views into one NSScrollView.
        - Ungroup
          Ungroups the contained views.
    - Disable Guideline
      This item toggles between Enable Guideline and Disable Guideline. This allows the user to turn on or off the guides which appear when placing views in a window or view in Gorm.
    - Font Panel The Font Panel allow you to modify fonts of your views.
- Classes
  Contains menus for working with classes.
    - Create Subclass
      Creates a subclass of the currently selected class in the current document classes view.
    - Load Class
      Loads a class from a .h file into the current document.

    - Create Class Files
      Generates a .h and .m file from the currently selected class in the current document classes view.
    - Instantiate
      Creates an instance of the selected class in the current document classes view.
    - Add Outlet/Action
      Adds an outlet or an action depending on what is selected in the document classes view. If the outlet icon is selected, it will add an outlet, if it the action icon is selected it will add an action.
    - Remove
      Removes the currently selected outlet, action or class.
- Tools
  Contains the inspector and the palette menus
    - Inspector
      Shows the inspector
    - Palette
      Shows the palette
    - Load Palette
      Opens a file panel and allows the user to load a palette into Gorm.
- Layout
  Contains a menu for working with alignement and layout of you views
    - Alignement Wich produces a submenu
        - Center Vertically
          Center Vertically two or more views. TODO :explain what is the reference view
        - Center Horizontally
          Center Horizontally two or more views. TODO :explain what is the reference view
        - Left Edges
          TODO
        - Right Edges
          TODO
        - Top Edges
          TODO
        - Bottom Edges
          TODO
    - Bring to Front
      Bring to front the selected view
    - Send to Back
      Send to back the selected view
- Windows
  Shows currently open windows.

- Services
  Shows currently available services.
- Hide
  Hides the application.
- Quit
  Quits the application.

5 Implementation

5.1 Notes on implementation

The IB documentation on how object selection is managed and how editors and inspectors are used is unclear ... so I've gone my own way.

1. When a document is loaded, the document object creates an editor attached to each top-level object in the user interface (NSMenu and NSWindow objects).

These editors must be aware of their edited objects being clicked upon, and clicking on one of these should cause the corresponding editor to become the active editor.

The active editor is responsible for handling selection of the edited object (and any objects below it in the object hierarchy). Upon change of selection, the editor is responsible for sending an IBSelectionChangedNotification with the selection owner (normally the editor itsself) as the notification owner.

The main application watches for these notifications in order to keep track of who has the selection.

5.2 Connections

The connection API is the same as that for IB, but with the extension that the document object must implement [-windowAndRect:forObject:] to return the window in which the object is being displayed, and the rectangle enclosing the object (in window base coordinates).

This information is needed by Gorm so that it can mark the connection.

The editors mananging the drag-and-drop operation for a connection must call '`[NSApp -displayConnectionBetween:and:]`' to tell Gorm to update its display. This method sets the values currently returned by '`[NSApp -connectSource]`' and '`[NSApp -connectDestination]`'.

6 Preferences

The preferences panel contains a number of useful customizable options which can be used to modify the behavior of Gorm.

Some of these defaults can be safely modified from the command line by the user.

- PreloadHeaders
  The user can define a set of headers to load when Gorm starts creation of a new .gorm file. This is useful when the user is building a framework or a set of interfaces for a large application.
- ShowInspectors
  Controls whether the inspector shows when Gorm is started.
- ShowPalettes
  Controls whether the palettes window shows when Gorm is started.
- BackupFile
  Determines if the old .gorm is moved to .gorm~ when the modified version is saved.
- AllowUserBundles
  If the user sets this to YES, they will still get a warning, but Gorm won't quit.

7 Basic Concepts

This chapter will explain some of the basic things you need to understand before starting work on a new application.

7.1 Getting Started

First you need to understand a few basic concepts. Gorm's main window includes a few standard entries which must be explained before we can proceed.

They are:

- NSOwner
- NSFirst
- NSFont

7.2 What is NSOwner?

NSOwner is the class which "owns" the interface. This is, by default, NSApplication, but it can be any class you like. You can change it by selecting NSOwner in the document window and going to the "Custom Class" inspector in the inspectors window. From there, you should see all of the classes which the NSOwner can assume. We'll discuss more about this later when we go over how to create a new application

7.3 What is NSFirst?

NSFirst is your interface to the responder chain. NSFirst is representative of the current "first responder" in the application. When you want a message, such as a changeFont: message, to go to the current first responder from, say, a menu, you connect the menu item to the NSFirst object in the document window. By doing this, it means that whichever object has first responder status at that time in the application will become the reciever of the "changeFont:" message.

7.3.1 Responders

A responder is any subclass of NSResponder. This includes NSWindow, NSView and all of the NSControl subclasses.

7.3.2 The Responder Chain

The responder chain is a sequence of objects which are called to determine where a message sent to the first responder will go. A message invoked on the first responder will be invoked on the first object in the responder chain which responds to that message.

The object which this message will be called on is determined in the method [NSApplication targetForAction:]. The call sequence is as follows, it will only proceed to the next step in each case if the current step fails to respond to the message which was invoked:

- The firstResponder of the keyWindow, if one exists.
- Iterates through all responders by pulling each in the linked list of responders for the key window.
- It then tries the keyWindow.

- Then the keyWindow's delegate
- if the application is document based it tries the document controller object for the key window.
- then it tries the mainWindow's list of responders (as above)
- the mainWindow's delegate
- if the app is document based, it tries the document controller for the main window
- and finally, it tries the NSApplication delegate.

If all of the options in this list are exhausted, it then gives up and returns nil for the object which is to respond.

7.4 What is NSFont?

NSFont represents the NSFontManager object for the application. This object is a shared singleton. This means that, for any given app, there should be only one instance of the object. This object is generally added to the document window when another objec, such as a Font menu item, is added to the interface, which, in turn, requires that this object be added to the document.

7.5 The awakeFromNib method

This method is called on any custom object which is unarchived from a nib/gorm file. This method is called on all objects after the entire archive has been loaded into memory and all connections have been made. Given all of this, you should not make any assumptions at all about which objects have been called and which have not. You should not release any objects in this method.

8 Creating an Application

If you have ProjectCenter, you need to open it and create an "Application" project. Create it with the name "FirstApp". From there you can open the MainMenu.gorm by clicking on interfaces and selecting MainMenu.gorm. If Gorm.app is properly installed, you Gorm should start up.

If you don't have ProjectCenter, you can create the Gorm file by hand. First you need to start Gorm. You can either do this by doing '`gopen Gorm.app`' from a command line prompt, or you can invoke it from the Dock or from the workspace's file viewer.

You then need to select the '`Document`' menu, and then '`New Application`'. This should produce a new document window, with a menu and an empty window. This should be the same as with the ProjectCenter gorm file since this is the basic starting point for an application.

For the sections below... only do one or the other, not both.

8.1 Creating A Class In Gorm

There are two ways to do this next operation. I will take you through each step by step. First click on the classes icon in the toolbar on the top of the Gorm document window. You should see the view below change to an outline view containing a list of class names. Once this happens we're ready to create a class. Select the class you wish to subclass in the outline view. For our example we will use the simplest: NSObject. Select it by clicking on the class name once. Then go to the Classes menu in the main menu and select Create Subclass (you can also type Alt-Shift-c, which will do this as well. The new class will be created in the list with the name "NewClass".

8.2 Using The Outline View

From here double click on the subclass name to make it editable. Type the name of the class and hit enter. For our example, please use the class name MyController. When you hit enter an alert panel will appear and warn you about breaking connections, simply select OK and continue.

This method of inputting the classes was inspired by IB in OPENSTEP 4.2/Mach which had functionality very similar to this. For users of that the transition to Gorm will be seamless.

8.2.1 Adding Outlets In The Outline View

Too add an outlet, select the round icon with the two horizontal lines in it (it sort of looks like a wall outlet. This should become depressed. Here you need to go to the Gorm Menu, under Classes select "Add Outlet/Action". Each time you press this menu item another outlet will be added with a name similar to newOutlet, as you add more the number at the end will increase. For now add only one outlet.

To rename the outlet simply double click it and change it's name like you did the class above to "value" for the sake of our example.

8.2.2 Adding Actions In the Outline View

The procedure to add on action is precisely the same as adding an outlet, except you must click on the button which looks like a target (a circle with a + inside). Add an action and name it "buttonPressed:" for the sake of our example.

8.3 Using The Class Edit Inspector

This way is much more inline with the "OPENSTEP/GNUstep" philosophy. For each object there is an inspector, even for Class objects.

Once you have created the class as described in the previous section "Creating a Class In Gorm", you must skip to this section to use the inspector. In the Gorm main menu select Tools and then select "Inspectors". This will make certain that the inspectors window is being displayed. Once the inspectors window is up move the pulldown on the top to "Attributes" and select the class you created which should, at this point, have the name "NewClass". You'll notice that the "Class" field at the top which shows the name's background color has turned white, instead of grey. This indicates that this class name is editable. Erase "NewClass" from the text field and type "MyController".

8.3.1 Adding Outlets In The Inspector

Adding outlets is very intuitive in the inspector. Simply select the "Outlets" tab in the tab view and click "Add" to add more outlets, and "Remove" to remove them. For the sake of our example, add one outlet and name it "value".

8.3.2 Adding Actions In the Inspector

Very much like above only with the "Actions" tab, add an action called button pressed.

8.4 Instantiating The Class

In the Classes outline view select the new class you've created, now called MyController and then go to the Gorm menu and select Classes, and then Instantiate. The document window should shift from the classes view to the objects view. Amoung the set of objects should be a new object called MyController.

8.5 Adding Controls from the Palette

Go to the Gorm menu and select Tools, then Palettes. This will bring the palette window to the front. The second palette from the left is the "ControlsPalette". Select that one and find the button object (it should have the word "Button" in it). Drag that to the window and drop it anywhere you like.

Repeat this operation with the text field. It's the control with "Text" in it. We are now ready to start making connections between different objects in the document.

8.5.1 Making Connections

The type of application we are creating is known as a "NSApplication delegate" this means that the MyController object will be set as the delegate of NSApplication.

To make this connection click on NSOwner and hold down the Control button, keep it pressed as you drag from the NSOwner object to the MyController object. The inspectors

window should change to the Connections inspector and should show two outlets "delegate" and "menu". Select the "delegate", at this point you should see a green S and a purple T on the NSOwner and MyController objects respectively, and press the "Connect" button in the inspector. In the "Connections" section of the inspector you should see an entry which looks similar to "delegate (MyController)" this indicates that the connection has been made.

Now we need to make connections from the controller to the textfield and from the controller to the button. Select the MyController object and Control-Drag (as before) from the object to the text field, this will make an outlet connection. You should see the connections inspector again, this time select the "value" outlet and hit Connect.

Next, control-drag from the button to the controller, this will make an action connection. The connections inspector should again appear. This time you need to select the "target" outlet, to get the list of actions. The list should have only one entry, which is "buttonPressed:" since this is the one we added earlier. Press Connect. You should see an entry like "buttonPressed: (MyController" in the Connections section of the inspector.

It is also possible to make this connection to NSFirst, but to keep things simple, make it directly to the object. If you make the connection to buttonPressed: on NSFirst the functionality of the application will be unchanged, but the invocation will take the path described above in the section which describes "The Responder Chain".

8.6 Saving the gorm file

At this point you must save the .gorm file. Go to the Gorm menu and click Documents and then select "Save". If the document was opened from a pre-existing .gorm, it will save to that same file name. If it is an UNTITLED .gorm file a file dialog will appear and you will need to select the directory where you want to store the .gorm file and type the name of the .gorm file.

8.7 Generating .h and .m files from the class.

This is different than saving, some people have gotten this confused with the idea of Gorm generating the code for the gui. Gorm does nothing of the sort (grin).

Go to the Classes section in the Document window and select the MyController class yet again. Now go to the Gorm menu and select Classes and the select "Create Class Files". This will bring up a file panel and it allow you to select the directory in which to put the files. It will first create the MyController.m file and then the MyController.h file. Simply select the directory in which your app will reside and hit okay for both. You can change the names, but the default ones, which are based on the class name, should be sufficient. When you look at the .m for this class, you should see the '`buttonPressed:`' method with the commecnt '`/* insert your code here */`' in it. Delete this comment and add '`[value setStringValue: @''Hello''];`'. The class should look like this after you're done:

```
/* All Rights reserved */
#include <AppKit/AppKit.h>
#include "MyController.h"
@implementation MyController
- (void) buttonPressed: (id)sender
```

```
{
[value setStringValue: @"Hello"];
}
@end
```

You recall, we connected the textfield to the "value" variable. The call above causes the method setStringValue to be invoked on the textfield you added to the window.

Also, note that the name of the method is "buttonPressed:". This is the action which is bound to the button. When it is pressed the text in the textfield should change to "Hello".

You now need to build the application either by copying in a GNUmakefile and making the appropriate changes or by using ProjectCenter's build capability, depending on if you use it or not.

This app is available as "SimpleApp" in the Examples directory under the Documentation directory distributed with Gorm. Hopefully this has helped to demonstrate, albeit on a small scale, the capabilities of Gorm. In later chapters we will cover more advanced application architectures and topics.

9 Another Simple Application

This chapter will describe an application, very much like the previous one, but using a slightly different structure. This application builds on the previous application and uses WinController as the NSOwner of the app instead of making it the delegate of NSApplication.

9.1 Adding Menu Items

Select the first palette in the palette window, this should be the MenusPalette. The palette will have a bunch of pre-made menu items on it that you can add. We want to keep this simple, so grab the one called "Item" and drag it over to the menu in main menu nib (the menu on the screen, not the one in the objects view). As you have this object over the menu, the copy/paste mouse cursor should appear (it looks something like one box over another box at a 45 degree angle). Where you drop the menu determines it's position in the menu. You can always drag it to a new position after you've placed it by simply selecting and dragging up or down. Once you've placed the menu item, double click on the title and change it to "Open"

You can also change the name in the NSMenuItem attributes inspector. Now you must add openWindow: to MyController and make the connection from the "Open" menu item to NSFirst. In the connections inspector, find the "openWindow:" action. You could simply make the connection directly, but this is an exaple to show you that this connection will work as well. Whichever object has First Responder status will be tested to see if it responds to this method.

The implementation for openWindow: in MyController should simply be:

```
- (void) openWindow: (id) sender
{
winController = [[WinController alloc] init];
}
```

Also add the winController attribute and an include to allow WinController to be referenced in the MyController.m file.

9.2 Making a Controller-based .gorm file

Create a new .gorm file as described in the previous section using the "New Module" menu item. Under "New Module" select "New Empty". This should produce a .gorm file with only NSOwner and NSFirst. From the WindowsPalette (which should be the second palette in the palette window) drag a window to the location where you want it to appear on the screen. In the window add a button called "Close".

Go through the same steps you went through previously to create MyController, except for adding the outlets/actions, but this time with the name WinController. Add an outlet called window and an action called "closeWindow:".

Now, instead of instantiating the class go back to the objects view and select the NSOwner object. After that select the "Custom Class" inspector. Look for the entry for WinController and select it. You now must connect the "window" outlet to the Window you added previously.

Switch back to the objects view, then Control-Drag not to the window on the screen, but to the window's representation in the objects view. In the connection inspector select the window outlet and click Ok.

Save the .gorm as using the name Controller.gorm in the project directory.

Generate the Controller.h and Controller.h files as described in the previous section.

9.2.1 Add the init method to WinController

Add an implementation of the action "closeWindow:" to WinController and also an init which loads the gorm/nib file and declares itself as the owner. Here's how:

```
/* All Rights reserved */
#include <AppKit/AppKit.h>
#include "WinController.h"
@implementation WinController
- (id) init
{
if((self = [super init]) != nil)
{
if([NSBundle loadNibNamed: @"Controller" owner: self] == NO)
{
NSLog(@"Problem loading interface");
return nil;
}
[window makeKeyAndOrderFront: self];
}
return self;
}
- (void) closeWindow: (id) sender {
[window close];
}
- (void) dealloc { [super dealloc]; RELEASE(window); }
@end
```

The Controller gorm will be loaded and the connections will be made to the current instance, i.e. window will point to the window object instantianted in the .gorm file and all actions declared in the .gorm file which are attached to the object NSOwner will be resolved on self.

9.3 Running the App

Type the command '`open Controller.app`' on the command line in the project directory. Once the application has started it should look very much like the first application. Select the "Open" button from the Menu and you should see the second window pop up, now choose close, this will call the method "closeWindow:" which should cause the window to disappear.

10 Advanced Topics

This section will cover some topics which won't be of general interest to most users. The details in this section pertain to the internal workings of Gorm.

10.1 Gorm file format

The current Gorm file format is basically just a set of objects, encoded one after another in a continuous stream with some markers indicating when a new class starts or which class is encoded.

10.1.1 The Name Table

Each object in the .gorm file has a name assigned to it by the application. This allows Gorm to refer to the objects by a name once they are loaded rather than an address. Each name is associated with it's object in a dictionary which preserves the overall structure of the GUI which has been created.

10.1.2 The Custom Class Table

This is only used when the user has associated a custom class with an existing instance in the gorm file. If the user has, for instance, added an NSWindow to the gorm, he/she can use the custom class inspector to select a subclass of NSWindow to change to.

10.1.3 Connections Array

This array is used to form the connections after the .gorm file is loaded. The method '`[... establishConnection]`' is never called on either NSNibControlConnector or NSNibOutletConnector objects while in Gorm. This prevents the connections from having any effect while they are being edited in Gorm itself. Once they are loaded, the establishConnection method is called and the connections are made.

10.2 Custom Class Encoding

Custom objects are an interesting challenge in Gorm. By definition, custom classes are not known to Gorm, unless they are in a palette (covered elsewhere). For classes which are not in a palette instances of these classes in Gorm are encoding in one of three ways:

- A Proxy - This is a standin object which takes the place of the custom object. This is usually used when the superclass of the object is a non-graphical object, such as a controller. The init message is called on this object when it's unarchived.
- A Custom View - This is a standin view object similar to the one descrribed above, but it is a subclass of NSView. When this is used the initWithFrame: message is called on the view instance which is created (based on what view subclass the user selects)
- A Template - Probably the most interesting of the three. This is a standin class which uses an existing instance created in Gorm to build a custom subclass from. For instance when a window subclass is created, call it MyWindow, a template class called GSWindowTemplate is used to hold the NSWindow created in Gorm as well as the name of the subclass to be created when the class is unarchived outside of Gorm as well as some additional information. When the classes are unarchived in the running

app, the designated initializer for that class will be invoked, except in the case of NSControl subclasses. See the Apple documentation for more information.

All custom instances have awakeFromNib invoked on them when they are unarchived from the .gorm file. This allows the user to do whatever additional setup that needs to be done, such as setting attribute. Classes which are "known" are, of course, directly encoded into the .gorm file.

10.2.1 Restrictions On Your Custom Subclasses

The restrictions here are the same as those in Apple's InterfaceBuilder. In general, you cannot have additional information which is expected to be decoded in an initWithCoder: method from a custom class which uses one of the methods in the previous section. This is because, by definition, Gorm doesn't know anything about these classes and allowing you to use them in Gorm in this way is a convenience to make it simpler for the developer. Gorm therefore, must use one of the proxies to encode the class since it cannot encode the class directly.

How can you get your classes into Gorm, you say? I'm pleased that you asked me that question. The best way to make your class known to Gorm so that you don't need to worry about the above restriction is to add a palette which contains your class. In this way, because you're literally linking the class into Gorm, you're making the class and it's structure known to Gorm so that it can encode the class directly. With the new palette loaded you can load and save classes containing real instances, not proxies, of your class encoded directly in the .gorm file. How to create a palette is discussed at length in the following section.

10.3 Palettes

10.3.1 Graphical Objects In A Palette

You are, by now, familiar with the built in palettes which are provided with Gorm. Palettes are a powerful feature which allows the developer to add his/her own objects to Gorm. It is possible for a developer to write custom inspectors, editors and palettes for use with Gorm. A good example of a custom palette is palettetest in the dev-apps/test in the GNUstep distribution. Assuming you don't have that, however, I will explain precisely what you need to do in order to create a simple palette. The entire process is very short and suprisingly simple. First open Gorm and selection Gorm->Document->New Module->New Palette. This will create a palette sized window. Once that's done go to the classes view in the main document window and find "IBPalette" in the class list. Create a subclass of that, the name can be whatever you want. For the purposes of our example we'll call it MyPalette. Drag a custom view to the window and choose the class you would like to add to the palette from one of your custom classes.

Once you've done this, generate the code for the classes (discussed in previous chapters). In the code, you'll add a method called "-(void) finishInstantiate" leave it empty for now. In the makefile for the palette make sure that the library or framework the view comes from is linked with the palette. Now build the palette.

After the palette is built you're ready to load it into Gorm. Go to the preferences panel and go to "Palettes". This should bring up a table view. Click on add. You should see a

open dialog open. Select the palette bundle with this. If the palette is successfully loaded, you should see the name appear in the list. One thing to note here. Once a palette is loaded, it can't be unloaded until you close and restart Gorm. This is because by loading the palette bundle, the code in the bundle is being linked into Gorm. This can't be undone, once it's done.

Now, you should see the palette in the set of palettes in the palette window. Simply scroll over to it and select it's icon. When you do this, you should see the view that you set up using the custom view displayed as an actual instance. Note that we used one of the techniques listed above, it is possible to use any of the three for any object you add to your palette. You can now drag the view from the palette to a new window.

10.3.2 Non Graphical Objects In A Palette

You may recall the creation of a method called "-(void) finishInstantiate" in the previous section. This section will make full use of that method. Re-open the palette you created before, but this time add an image view to the window. Then add to the image view, the icon you want to represent the non-graphical object. Here you'll need to add an ivar to the MyPalette class in both Gorm and in your source code called, imageView. Once you've done this make the connection between the image view and it's ivar.

Assuming that the class is called "NonUIObject", in finish instantiate, you'll need to add the following line of code:

```
id obj = [NonUIObject new];

[self associateObject: obj type: IBObjectPboardType with: imageView];
```

This code has the effect of associating the non-ui object with the ui object you just added to represent it. When you drag and drop the element which prepresents the object to something, it will copy the object, not the ui element, to the destination.

Congratulations, you now know how Palettes work.

11 Frequently Asked Questions

11.0.1 Should I modify the data.classes of file in the .gorm package?

My advice is never to do this, ever. Some have said that "they're plain text and I should be able to change them". My response to this rather loosely pronounced and weak rationale is that if they are modified I cannot and will not guarantee that Gorm will be able to read them or will function correctly if it does.

11.0.2 Why does my application crash when I add additional attributes for encoding in encodeWithCoder: or initWithCoder: in my custom class?

If you've selected the custom class by clicking on an existing object and then selecting a subclass in the Custom Class Inspector in Gorm's inspector panel, then when the .gorm file is saved, Gorm must use what is called a template to take the place of the class so that when the .gorm is unarchived in the running application, the template can become the custom subclass you specified. Gorm has no way of knowing about the additional attributes of your subclass, so when it's archived the template depends on the encodeWithCoder: of the existing class. Also, when AppKit loads the .gorm file, the initWithCoder: on the subclass is called to allow the user to do any actions, except for additional encoding, which need to be done at that time. This is particularly true when non-keyed coding is used, since, with keyed coding, it's possible to skip keys that are not present. The application may not crash if keyed coding is used, but Gorm would still not know about the additional attributes and would not be able to persist them anyway.

Please see information in previous chapters regarding palettes, if you would like to be able to add your classes to Gorm so that they don't need to be replaced by templates, or proxy objects.

11.0.3 Why does Gorm give me a warning when I have bundles specified in GSAppKitUserBundles?

Some bundles may use poseAs: to affect change in the existing behavior of some GNUstep classes. The poseAs: method causes an issue which may cause Gorm to incorrectly encode the class name for the object which was replaced. This makes the resulting .gorm file unusable when another user who is not using the same bundle attempts to load it.

11.0.4 How can I avoid loading GSAppKitUserBundles in Gorm?

You need to write to Gorm's defaults like this:

```
‘ defaults write Gorm GSAppKitUserBundles '()' '
```

Doing this overrides the settings in NSGlobalDomain for Gorm and forces Gorm not to load any user bundles at all. To eliminate this simply do:

```
‘ defaults delete Gorm GSAppKitUserBundles '
```

11.0.5 How can I change the font for a widget?

This is a simple two step process. Select the window the widget is in and then select the widget itself, then bring up the font panel by hitting Command-t (or by choosing the menu

item). By doing this you're making the window the main window and by selecting the widget, you're telling the editor for that object to accept changes. Then you can select the font in the panel and hit "Set". For some objects, the font panel isn't effective because those objects can't have a font directly set.

Concept Index