UML FOR OBJECTIVE-C

Excel Software
www.excelsoftware.com

Objective-C is a popular programming language for Mac OS X computers. The Unified Modeling Language (UML) is the industry standard notation for modeling object-oriented software. This paper shows how to model Objective-C software using UML.

Objective-C is one of several programming languages supported by Apple's Xcode development system. The Objective-C language is a superset of ANSI C with extensions for object-oriented programming. These extensions allow classes, categories and protocols to be defined, objects instantiated from classes and messages sent between objects.

UML defines a family of graphical notations for describing and designing software systems. It is an open standard controlled by the Object Management Group (OMG), an open consortium of companies. UML defines many diagram types including use cases that document user interactions with a software system, class models that show static class structure and relationships, state models that show how important events are handled and object models that show messages being passed between instantiated objects. This paper will focus on the class diagram, essential to any object-oriented design.

UML is a communication tool used by analysts, designers, programmers, testers, technical writers and managers. It provides the big picture of a software development project and helps to organize the contributions of each individual. UML provides a standard foundation that is often tailored to specific environments ranging from desktop applications to real-time, embedded systems and different programming languages like C++, Java and Objective-C.

This paper assumes a working knowledge of the Objective-C programming language and the UML modeling notation. The Xcode documentation includes a PDF document titled, The Objective-C Programming Language that describes the language, runtime system and grammar. Many books have been written on UML including a good introductory book titled **UML Distilled** by *Martin Fowler*.

After describing how Objective-C constructs can be mapped to UML, the MacA&D modeling tool is used to present class diagrams. MacA&D is a complete UML modeling tool with an Objective-C code generator. The example diagrams in this paper were auto-generated from Apple's example source code using the MacTranslator reengineering tool and presented within the MacA&D modeling tool. MacA&D and MacTranslator are available from Excel Software at www.excelsoftware.com.

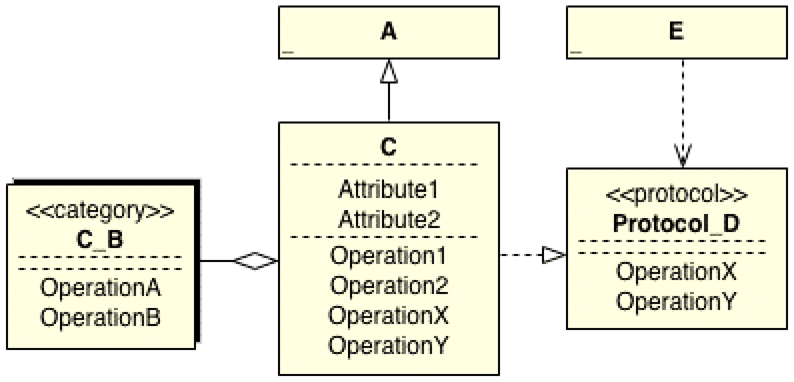
Map Objective-C Constructs To UML

There are three main constructs in the Objective-C language for grouping attributes and/or operations together, namely classes, categories and protocols. Normal classes are represented in the UML class diagram as a named box with dashed lines between a group of attributes names and a group of operation names.

A Category in Objective-C is an extension to an existing class. It adds methods to the original class that are inherited by all of that class's subclasses. A category cannot add attributes, just methods. It is represented in the UML class diagram as a class object with the <<category>> stereotype and highlighted with a shaded background. By convention, the name of a category class object indicates it base class. For example, the MyExtensions category of the NSArray class is named NSArray_MyExtensions.

Protocols in Objective-C can be represented on a class diagram with an interface object. An interface object looks just like a class with the addition of a <<protocol>> stereotype and the "Protocol_" prefix on its name. Since a class can use the same name as a protocol in an Objective-C program, the "Protocol_" prefix is a convention that provides a unique name for all protocol objects in the design model. In Objective-C, a protocol defines a collection of methods that can be implemented by any class. A protocol allows designers to create standardized interfaces that can later be implemented by some classes and used by other classes. Protocols have no attributes, just methods.

The UML class diagram below shows the object-oriented constructs in Objective-C represented by class objects and relationships.



UML Class Diagram for Objective-C Constructs

In the diagram above, the objects A, C and E represent normal classes. Class C has two attributes named Attribute1 and Attribute2 and four operations named Operation1, Operation2, OperationX and OperationY. Class A and E may also have attributes and operations, but we've chosen a suppressed presentation where they are hidden as indicated by the … in the lower left corner of the class box. The object named C_B is an Objective-C category of class C. Notice the shaded box outline and the stereotype <<category>>. The object named Protocol_D is an Objective-C protocol represented as a UML interface with the stereotype <<protocol>>.

In addition to various named boxes with attributes and operations, a UML class diagram connects nodes with lines indicating relationships between classes, categories and protocols.

Here are some important UML relationships between classes and how they're used to model Objective-C programs:

- The generalization relationship shows superclass and subclass pairs.
- The aggregation relationship shows a category is part of its parent class.
- The realization relationship shows how a class implements an interface.
- The dependency relationship shows how a class uses an interface.

In the diagram above, class A is a superclass of class C. Stated another way, subclass C inherits the attributes and operations of class A. This is represented in UML with a solid line from the subclass and hollow arrow pointing at the parent class.

Class C has an aggregation by reference to its category C_B. This is represented in UML as a solid line from the category object to a hollow diamond attached to class C. Think of the category as part of the C class, since other objects that communicate with an object instantiated from class C can use the new operations named OperationA and OperationB added by category C_B.

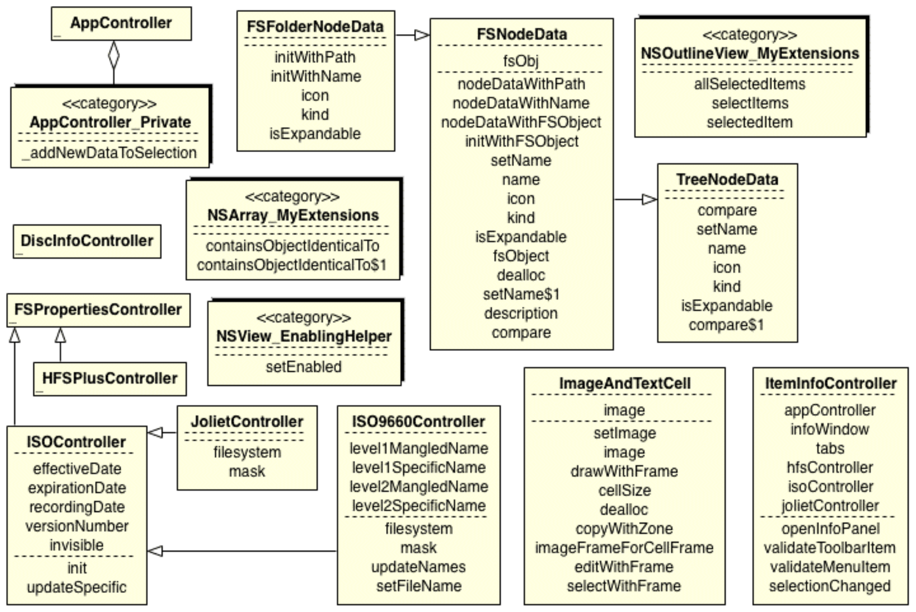
Class C implements OperationX and OperationY specified by protocol Protocol_D. UML shows that relationship with a dashed line and hollow arrow pointing at the implemented interface.

Class E depends on the interface specified by OperationX and OperationY in protocol Protocol_D. UML shows that dependency with a dashed line and open arrow pointing from E to Protocol_D.

UML Class Diagram of Objective-C Example Code

The static class structure of any Objective-C program can be illustrated in a UML class diagram with nodes representing classes, categories and protocols. Line styles and terminations represent important relationships.

As a more concrete example, consider the Objective-C example program provided with Apple's Xcode development system. The EnhancedDataBurn application presents a user interface to burn a data disc containing a complete virtual filesystem. That source code was scanned by MacTranslator, then imported into the MacA&D modeling tool. After a few minutes of cosmetic positioning, the diagram below was exported as an image file.



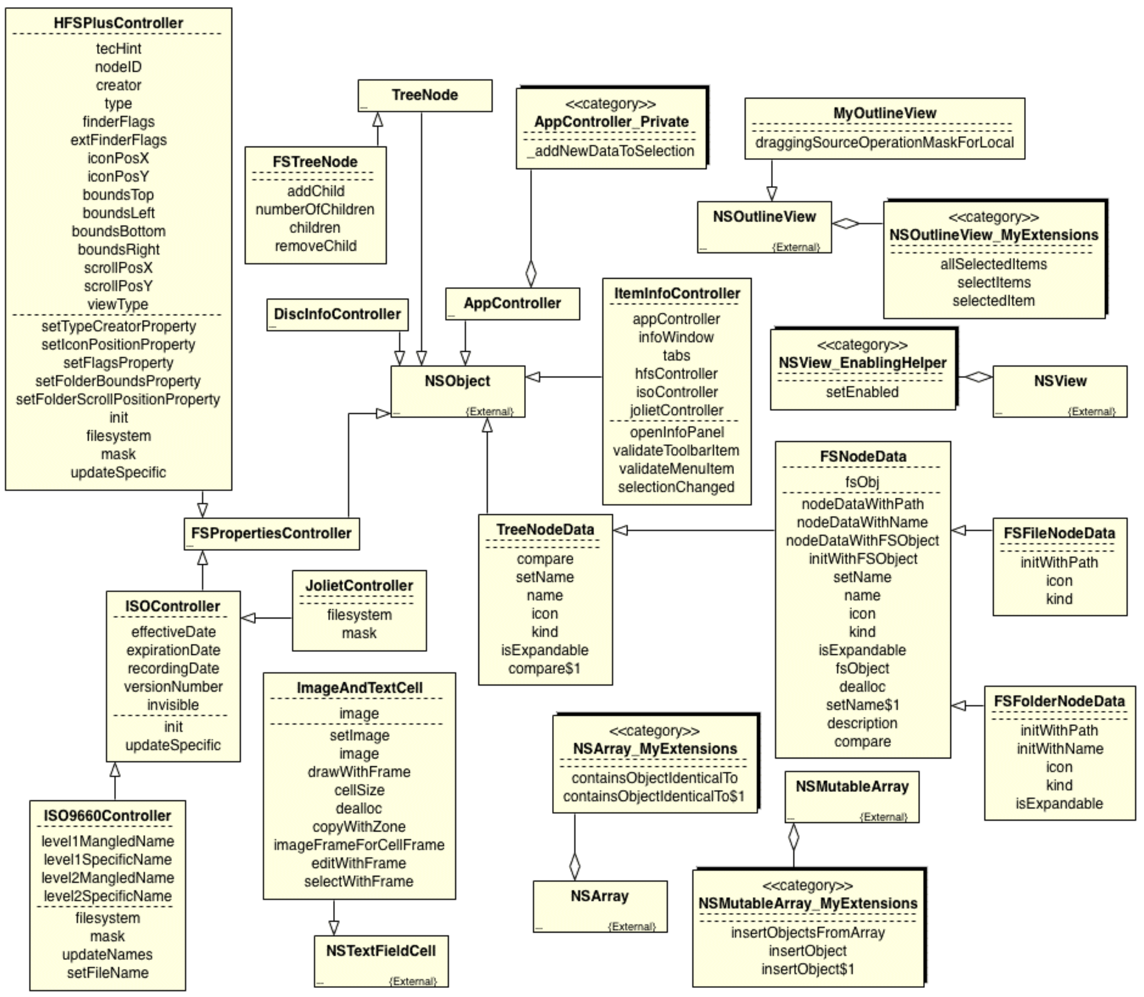
UML Class Diagram Generated from EnhancedDataBurn Example Objective-C Source Code

The diagram quickly reveals all classes and categories in the program. Categories are easily identified by the <<category>> stereotype. Inheritance relationships are shown between some of the classes. For example, the JolietController and ISO9660Controller class are subclasses of ISOController. The ISOController and HFSPlusController are subclasses of FSPropertiesController. The AppController class aggregates by reference its category AppController_Private.

Some classes and categories aren't linked to any other objects on the diagram. For example, NSArray_MyExtensions is a category of the NSArray class in the Cocoa framework. Many of the other standalone classes and categories are associated with framework classes that aren't shown on the diagram. Framework classes could have been included by also processing those interface files with MacTranslator.

Several of the larger classes having many attributes and operations have been suppressed as indicated by the … notation in the lower left corner of the class box. This was an arbitrary decision that can be changed during automated diagram generation or later within the diagram editor. There are many other options available to customize the diagrams and show different levels of detail. For example, the data types and access of attributes and operations and operation arguments can also be shown on the diagram.

Framework classes can also be shown on the diagram as external classes. MacTranslator collects a list of externally referenced classes that can be included in diagrams generated by MacA&D. Here is the EnhancedDataBurn diagram with external classes included.



UML Class Diagram for EnhancedDataBurn Example With External Classes Included

Dictionary Information

Depending on the context of a diagram and the level of detail that you want to reveal, some classes are shown in a suppressed view where attributes or operations are hidden. Classes can have a custom presentation that only shows important attributes or operations. A large project will likely have many class diagrams each showing one functional area of the project.

A class model graphically represents information stored in the underlying dictionary. When drawing class diagrams and typing information into MacA&D dialogs, the dictionary is being constructed behind the scenes. As a central repository of information, it allows the designer to add multiple instances of a class to different diagrams without retyping information. All details of a class are stored once in the dictionary regardless of how that class gets expressed on different diagrams.

The dictionary contains a separate entry for each class, class attribute and class operation. Each entry stores details about that item. For example, a class operation entry stores the operation's name, return data type, public or private access type, argument list, description, implementation notes, etc.

A visual inspection of the dictionary for a class entry shows that it stores information about relationships with other classes. Here is the simple dictionary syntax you'll see for these important relationships:

- ^ = Inheritance
- ! = Interface
- & = Aggregation

Inheritance is a cornerstone of all object-oriented software. It allows a subclass to inherit the attributes and operations of its superclass. The Interface relationship indicates an interface (protocol in Objective-C terminology) that a class implements. The Aggregation relationship identifies categories that extend the class.

The dictionary contains the summation of design information for the object-oriented software. It can be constructed indirectly through the process of drawing models and filling in dialogs. The dictionary can also be constructed automatically by scanning existing source code with MacTranslator and importing its output files into MacA&D. From dictionary information, class diagrams can be automatically generated.

Tools to Accelerate Development

Projects often require several people working together as a team to design, implement, test and manage the software development process. The MacA&D modeling tool provides structure to that process and supports the entire UML notation. Use it to define and manage use cases and requirements, draw models, generate Objective-C code, navigate from models to code, organize the testing effort and export design information as HTML reports or Microsoft Word documents that team members can use.

Not every project starts from a clean slate. Projects often port software from other platforms, other languages or add new features to existing legacy code. Even when you are starting from scratch you'll probably want to review and understand examples of existing source code or the work of others on the project. MacTranslator makes it easy to generate graphic UML class models and dictionary information from Objective-C source code. The process is fully automated and can usually be completed in minutes on unfamiliar code.