Functional Specification

Class Declaration Syntax and Run-Time Selector Mapping in the Objective-C Language

Requirements...based on perceived market need and technical dependencies.

- fix bugs related to foward references, compilation order, and incomplete class declaration files (or002, or017, or028, or042, or044, or058, or071, or072, or075, etc) .

- insure the integrity of the object module.

- allow classes to be dynamically linked (on systems like Apollo/NeXT/HP).

- eliminate language constructs that impede the development/management of applications involving many programmers, the language must be predictable and easy to use.

- provide a solid foundation for offering future enhancements to the language (e.g. static binding) and environment.

- ease integration of IC-paks developed independent of one another.

Specification

This section describes changes (from the users perspective) to the Objective-C language and runtime support system which meet the above requirements. The proposed strategy adds only one new construct to the language, a class declaration. It does, however, modify or replace several existing constructs. Note that the examples shown are for illustration purposes only.

(1a) Class Declaration Syntax

Changes:

- the developer would be required to provide the compiler with an interface declaration for the class. Private methods that are only used locally do not have to be listed.
- the developer would be required to *"#import"* the declaration for all of the classes it consumes. This facility will allow a more robust interface; the file that contains the class interface can also embody all the C constructs (pre-processor macros, user-defined types, typedefs) required to interface to the class.
- this replaces the current C_ and P_ files.

Example:

```
/* Expr.h: Objective-C compiler interface specification */
#import "Node.h"
#import "ConstValue.h"

#define declarePTR        id *_PTR = (id *)IV(self)

@interface Expr : Node
{
        id anExpr;
}
- decl;
- constValue;
@

@interface StropExpr : Expr
{
        id primary;
        id aToken;          /* "->", "." */
        id componentName;
}
+ initialize;
- decl;
- constValue;
@
```

(1b) Importing class interfaces

Changes:

- #import is a pre-processor extension defined by Objective-C. It differs from #include in two ways:

  (1) it will allow arbitrary nesting.
  (2) it will **not** #import a file more than once. This will free the user from having to use the following idiom (which is error prone):

  ```
  /* Fruit.h */
  #ifndef FRUIT.H
  #define FRUIT.H
  ...
  #endif FRUIT.H
  ```

- the *"= (Primitive, Demo)"* and *"= ClassA : ClassB(Primitive, Demo)"* constructs would be replaced by #import. They currently make selectors from the named message groups available during the translation of the current non-class/class source files.

- the "@requires" construct would also be replaced by #import. Currently, it helps the compiler manage the C_ and P_ file dependencies.

Example:

```
#import "Demo.h"
#import "Primitive.h"

main( argc, argv )
int argc;
char *argv[];
{
        id aFruit, anApple;

        aFruit = [Fruit create];
        [aFruit grow];
        anApple = [Apple create];
        [[[anApple color:"red"] flavor:"Macintosh"] diameter:7];
}
```

(1c) Class Definition Syntax

Changes (minor):

- specification of the superClass and/or instance variables is optional. If present, they must match the @interface declaration for the class. If they do not match, the compiler will issue a warning, indicating a module interface inconsistency.

- "@module" will replace the current start indicator ("=").
- "@" will replace the current finish ("=:").

Example

```
#import "Expr.h"

@module StropExpr : Expr
{
        id primary;
        id aToken;              /* "->", "." */
        id componentName;
}
+ initialize { ... };
- decl { ... };
- constValue { ... };
@
```

(1d) Conversion

To ease the conversion to this approach, the compiler will provide a *-genDecl:"aFile"* option. This option will create an interface declaration from the current class definition files.

Givin the command:

```
objcc StropExpr.m -genDecl:"StropExpr.h"
```

Input:

```
/* StropExpr.m: Objective-C compiler source */

@requires UserType, ConstValue, Decl;

= StropExpr : Expr(ObjectiveC, Collection, Primitive)
{
        id primary;
        id aToken;          /* "->", "." */
        id componentName;
}
+ initialize { ... };
- decl { ... };
- constValue { ... };
=:
```

Output:

```
/* StropExpr.h: Interface declaration for "StropExpr.m"
 * Produced by Objective-C on 1/17/87
 */
#import "UserType.h"
#import "ConstValue.h"
#import "Decl.h"
#import "Expr.h"
#import "ObjectiveC.h"    /* a group of related classes */
#import "Collection.h"    /* a group of related classes */
#import "Primitive.h"     /* a group of related classes */

@interface StropExpr : Expr
{
        id primary;
        id aToken;          /* "->", "." */
        id componentName;
}
+ initialize;
- decl;
- constValue;
@
```

(2) Runtime Selector Mapping

The additional support required for mapping selectors at runtime will have little impact on users of Objective-C that do not rely on the exact format of the structures generated by the compiler. Users that do rely on many of the internal details (e.g. id **_Classes) will have to convert to the new support structures that are detailed in the design notes on the technique the compiler will employ to accomplish this mapping.

- **@messages(<aPhylaList>) becomes obsolete.** It currently tells the Objective-C compiler to generate a global table of unique selector codes.

- @classes(<aClassList>) becomes obsolete. To prevent confusion, the facility for determining what order the classes are initialized must be preserved someplace else.