Design Notebook

Selector Mapping in the Objective–C Run–Time Support System.

Design Notes (evolving)...Implementation details on mapping selectors at runtime.

Selector Tables

Fruit.c

```
static char __msgSelectors[] = {
    'c','r','e','a','t','e',0,
    'c','o','l','o','r',':',0,
    'c','o','l','o','r',0,
    'd','i','a','m','e','t','e','r',':',0,
    'd','i','a','m','e','t','e','r',0,
    'g','r','o','w',0,
    'n','e','w',0,
};
```

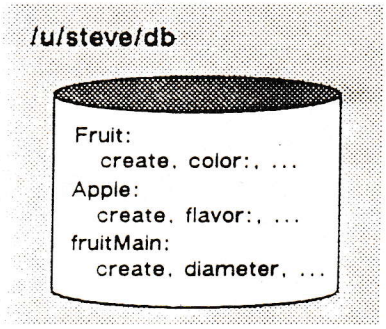
fruitMain.c

```
static char __msgSelectors[] = {
    'c','r','e','a','t','e',0,
    'd','i','a','m','e','t','e','r',0,
    'c','o','l','o','r',0,
    'g','r','o','w',0,
    'f','l','a','v','o','r',0,
    'c','o','l','o','r',':',0,
    'f','l','a','v','o','r',':',0,
    'd','i','a','m','e','t','e','r',':',0,
};
```

Apple.c

```
static char __msgSelectors[] = {
    'c','r','e','a','t','e',0,
    'f','l','a','v','o','r',':',0,
    'f','l','a','v','o','r',0,
    'f','l','a','v','o','r',':','d','i','a','m','e','t','e','r',':','c','o','l','o','r',':',0,
    'g','r','o','w',0,
    'n','e','w',0,
    'd','i','a','m','e','t','e','r',':',0,
    'c','o','l','o','r',':',0,
};
```

futures?



* each compilation unit (source/object file) that defines or references any selectors will contain its own local selector table.

* the combination of selector tables will be deferred until runtime.

* store type information (argument and return type) for each selector (not shown). Why?

  - the interpreter requires this information...it currently must read the P_* files.
  - the compiler can offer more sophisticated runtime support.

Dispatch Tables

Fruit.c

```
static struct _SLT _clsFruit[1]={
   (SEL)&__msgSelectors[0], (id (*)())_1_Fruit, /* create */
};
static struct _SLT _nstFruit[5]={
   (SEL)&__msgSelectors[7],  (id (*)())_2_Fruit, /* color: */
   (SEL)&__msgSelectors[14], (id (*)())_3_Fruit, /* color */
   (SEL)&__msgSelectors[20], (id (*)())_4_Fruit, /* diameter: */
   (SEL)&__msgSelectors[30], (id (*)())_5_Fruit, /* diameter */
   (SEL)&__msgSelectors[39], (id (*)())_6_Fruit, /* grow */
};
```

Apple.c

```
static struct _SLT _clsApple[1]={
   (SEL)&__msgSelectors[0], (id (*)())_1_Apple, /* create */
};
static struct _SLT _nstApple[4]={
   (SEL)&__msgSelectors[7],  (id (*)())_2_Apple, /* flavor: */
   (SEL)&__msgSelectors[15], (id (*)())_3_Apple, /* flavor */
   (SEL)&__msgSelectors[22], (id (*)())_4_Apple, /* flavor:diameter:color: */
   (SEL)&__msgSelectors[45], (id (*)())_5_Apple, /* grow */
};
```

* each entry will be mapped to a unique code at runtime.

* no need for the compiler to use an extra level of indirection when initializing the dispatch tables. Eliminate msgImpFind.**fixAllDispatchTables(id * msgFirstId)** and **fixDispatchTable(SHR cls)**. The structure template "struct __SLT { char **_cmd; ... }" is therefore obsolete.

* the structure tag used to identify the class and meta-class dispatch tables should include the class name – so that we can move towards lifting the one class per source file restriction.

Reference Tables

Fruit.c

```
static SEL _selRefs[] = {
   (SEL)&__msgSelectors[44],   /* new */
   (SEL)&__msgSelectors[7],    /* color: */
   (SEL)&__msgSelectors[20],   /* diameter: */
};
```

fruitMain.c

```
static SEL _selRefs[] = {
   (SEL)&__msgSelectors[0], /* create */
   (SEL)&__msgSelectors[7], /* diameter */
   (SEL)&__msgSelectors[16], /* color */
   (SEL)&__msgSelectors[22], /* grow */
   (SEL)&__msgSelectors[27], /* flavor */
   (SEL)&__msgSelectors[34], /* color: */
   (SEL)&__msgSelectors[41], /* flavor: */
   (SEL)&__msgSelectors[49], /* diameter: */
};
```

Apple.c

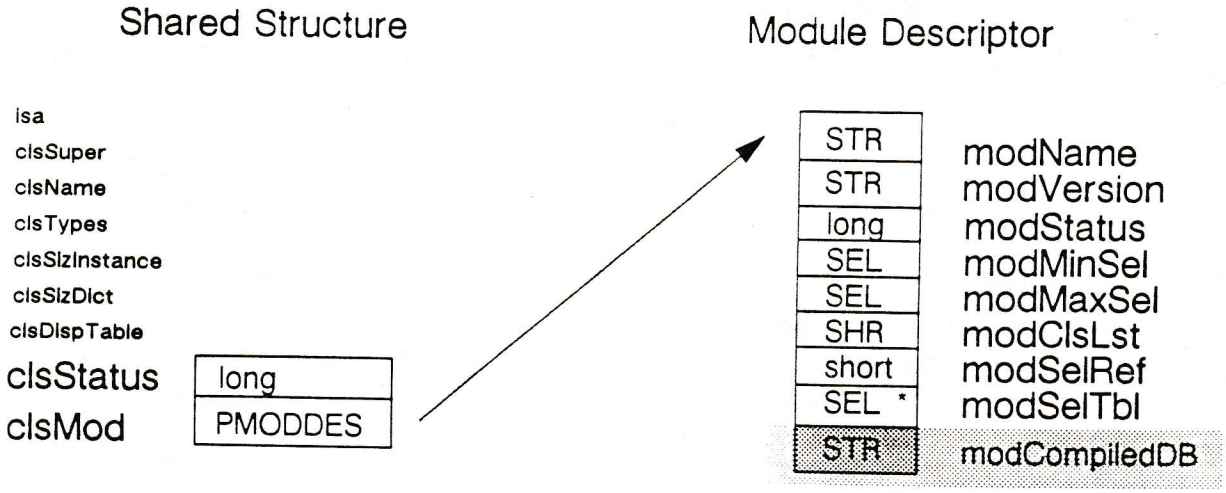
```
static SEL _selRefs[] = {
   (SEL)&__msgSelectors[50], /* new */
   (SEL)&__msgSelectors[22], /* flavor:diameter:color: */
   (SEL)&__msgSelectors[7], /* flavor: */
   (SEL)&__msgSelectors[54], /* diameter: */
   (SEL)&__msgSelectors[64], /* color: */
};
```

```
_msg(anId, _selRefs[0 <= n < _nSelRefs]);
```

* each compilation unit that sends messages will contain its own **local** reference table, this replaces the current Phyla tables which are published to the entire application.

* each entry will be mapped to a unique code at runtime.

* the message dispatcher does **not** need to be modified.




```
#define CLS_MAPPED              0x1L

#define CLS_FACTORY             0x2L
#define CLS_META                0x4L

#define CLS_BEINGINITIALIZED    0x8L
#define CLS_INITIALIZED         0x10L

#define CLS_POSING              0x20L
```

```
#define MOD_MAPPED      0x1L
#define MOD_EARLYMAP    0x2L

#define MOD_INITIALIZED 0x4L
```

* have the compiler generate a "module descriptor" for each module that defines/references any selectors.

changes to "struct _SHARED":

* add a pointer to the module descriptor that represents the object module where the class is defined.

* add a status word. CLS_MAPPED will indicate the classes dispatch tables have been mapped to unique codes. Notice, this has general utility.

* consider storing instance variable names (.clsInstNames?), not only types (.clsTypes). Why?

  - the interpreter requires this information...it currently must read the C_* files.
  - the compiler can offer more sophisticated runtime support. [**anObject show**] could provide much more useful information.

CompileTime

Fruit.c

```
static struct modDescriptor _modDesc = {
    "Fruit",
    "objcc v3.7",
    0L,
    (SEL)&__msgSelectors[0],
    (SEL)&__msgSelectors[44],
    &_Fruit,
    3,
    _selRefs
};
struct modDescriptor *_BIND$Fruit()
{
    return &_modDesc;
}
```

Apple.c

```
static struct modDescriptor _modDesc = {
    "Apple",
    "objcc v3.7",
    MOD_EARLYMAP,
    (SEL)&__msgSelectors[0],
    (SEL)&__msgSelectors[64],
    &_Apple,
    5,
    _selRefs
};
struct modDescriptor *_BIND$Apple()
{
    return &_modDesc;
}
```

fruitMain.c

```
static struct modDescriptor _modDesc = {
    "fruitMain",
    "objcc v4.0",
    0L,
    (SEL)&__msgSelectors[0],
    (SEL)&__msgSelectors[49],
    (SHR)0,
    8,
    _selRefs
};
struct modDescriptor *_BIND$fruitMain()
{
    return &_modDesc;
}
```

LinkTime

__linkVector.c

```
#include "module.h"

PMODDES _BIND$Fruit();
PMODDES _BIND$Apple();
PMODDES _BIND$fruitMain();

static struct modEntry {

    PMODDES (*modLink)();
    PMODDES modInfo;

} _modControl[] = {

    _BIND$Fruit,     0,
    _BIND$Apple,     0,
    _BIND$fruitMain, 0,

    0, 0
};

PMOD _objcModules = _modControl;
```

This file would be produced by a "post-linker" that would be part of the Objective-C compiler tools chain.

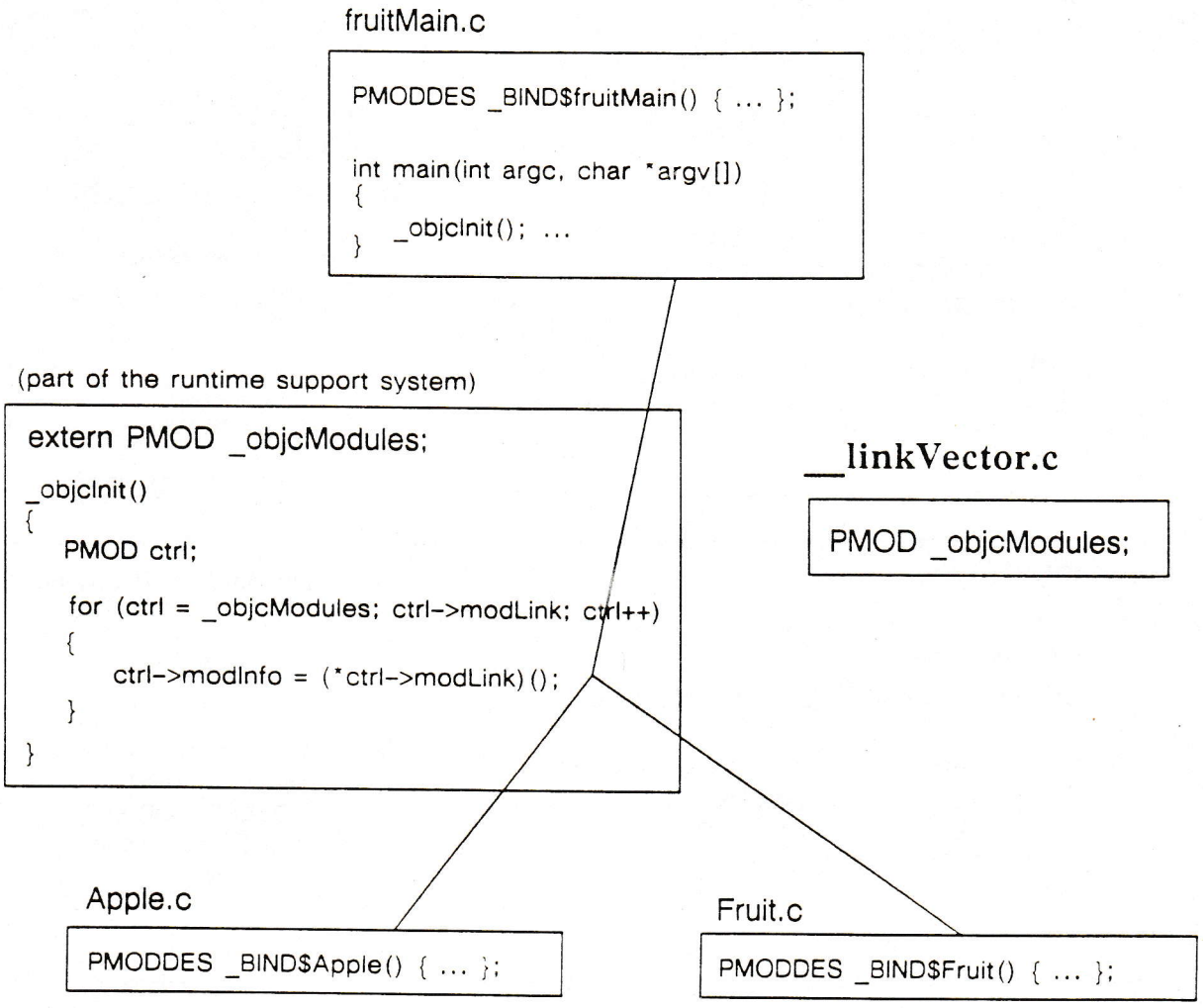
It would then be included as part of the application.

* the compiler must be taught to generate a _BIND$* entry point for any source file that defines or references any selectors. This entry point will be used to describe information about the module to the Objective-C runtime support system.

* replaces/automates "facilities" provided by the **@classes(ClassA, ClassB)** and **@messages** constructs.

* the control driver must be modified to incorporate the post link operation. This implies that applications written using Objective-C must be linked using the Objective-C control driver...(currently some "make" files might assume they can invoke the linker directly, this will no longer be acceptable).

RunTime Initialization



* the compiler must generate a call to **_objcInit** as the first statement in main(). This implies that the source file that contains "main()" must be compiled using Objective-C. We could rely on the linker (ld -e _objcInit) to accomplish this, however this is more likely to cause port problems?.?

* once this initialization has finished, the runtime system will have all the information it needs to map selectors dynamically, as messages are sent.

* **this has general utility!**

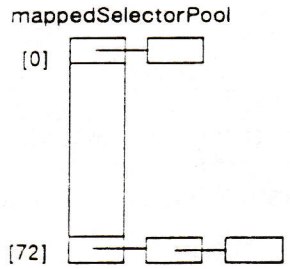
* makes future enhancements to the runtime support system less painful.

RunTime Selector Mapping

mapClass(SHR cls);
mapSelector(STR *sel);
mapSelRefs(PMODDES desc);

fruitMain.c

```
int main(int argc, char *argv[])
{
    _objcInit();
    aFruit = _msg(Fruit, _selRefs[0]);
    ...
}
```




```
_objcInit()
{
    PMOD ctrl;
    PMODDES desc;

    for (ctrl = _objcModules; ctrl->modLink; ctrl++)
    {
        desc = ctrl->modInfo = (*ctrl->modLink)();
        if (desc->modClsLst && mod_EARLYMAP(desc)) {
            mapClass(desc->modClsLst);
            mapSelRefs(desc);
        }
    }
}
```

```
_msgImpFind( SHR cls, SEL sel, ... )
{
    .
    .
    .
    if (!ISMAPPED(cls))
        mapClass(cls);
    if (!ISSELECTOR(sel))
        mapSelector(&sel);
    .
    .
    .
}
```

initialization — when? — message dispatch

* _msgImpFind() must change; since the interface will remain the same this should not affect anything.

* maintain a pool of mapped selectors at runtime. Selectors will cease to be merely a pointer to an array of characters, they will be a unique identifer (1.._maxSelector) determined by the runtime support system.

* provide a method to map classes on demand to bridge the gap between initialization time and message dispatch time.

Objective-C language constructs that are affected.

1. @messages - Tells the Objective-C compiler to generate a global table of unique selectors (codes).

```
static char __msgSelectors[] = { ... };
char *_minSelector= &__msgSelectors[0], *_maxSelector= &__msgSelectors[1369];
char *GroupA[] = { ... };
char *GroupB[] = { ... };
char *Primitive[] = { ... };
static char **__phylaTable[] ={
            GroupA,
            GroupB,
            Primitive,
            0
};
char ***_phylaTables = __phylaTable;
int _nPhyla = 3;
```

3. @classes(A,B)

Produces

```
extern struct _SHARED
            _A,
            _B,
            _Object;

// The factory objects belong with the object module for the class...
id          A = (id)&_A,
            B = (id)&_B,
            Object = (id)&_Object;

// the order of classes in this table "partially" determines the order in which classes are initialized.
static id *_clsTable[] ={
            &A,
            &B,
            &Object,
            0
};
id **_Classes = _clsTable; // modified by __insertClass(SHR *myClass)
int _nClasses = 3;
```

Research getting rid of this construct, without eliminating the concept of a class table. It would be nice to store A with the object module for the class.

2. @selector - This provides a compile-time way to get a handle on THE unique representation of a selector string. The selector code, if you will. This mechanism will now use the "mapSelector()" facility to obtain the unique representation, rather than rely on the global pools in the previous system.

Idiom

```
IMP cltnAtPut = [cltn methodFor:@selector(at:put:)];
(*cltnAtPut)(cltn, @selector(at:put:), (n), (obj));
```

Produces

```
IMP cltnAtPut  = (*(id (*(*)())())_msg)(cltn,Primitive[40]/*methodFor:*/,Primitive[61]);
(*cltnAtPut)(cltn, Primitive[61], (n), (obj));
```

6. @requires <classList> ;

Helps Objective-C manage lateral dependencies by writing this list to the "C_" file that represents the class being compiled. This is required mostly as a side-effect of present class and message group management. If the compiler stops managing these files automagically, perhaps this will become obsolete.

Usage

```
@requires Apple, Fruit;
```

Produces

```
extern id Apple, Fruit;
```

Objective-C runtime support macros/functions that are affected.

1. SEL, ISSELECTOR(sel), cvtToSel(STR aString)

Object.m:

```
+ (BOOL)instancesRespondTo:(STR) aSelector {
        if (!ISSELECTOR(aSelector))
                aSelector = (*_cvtToSel)(aSelector);
        if(aSelector == 0) // not a valid selector
                return NO; // quick answer
}
- (BOOL)respondsTo:(STR)aSelector { ... }
- perform:(STR)aSelector { ... }
- perform:(STR)aSelector with:anObject { ... }
- perform:(STR)aSelector with:obj1 with:obj2 { ... }
- doesNotRecognize:(STR)aMessage { ... }
- (IMP)methodFor:(SEL)aSelector { ... }
+ (IMP)instanceMethodFor:(SEL)aSelector { ... }
```

IdArray.m, Cltn.m, SortCltn.m:

```
/*
 *      Objects that have selectors as instance variables that get
 *      filed out via the "AsciiFiler" present problems. For example:
 */
= SortCltn : BalNode (Collection, Primitive)
{
        SEL cmpSel; // selector to use for comparing
}
- elementsPerform:(SEL)aSelector { ... }
- elementsPerform:(SEL)aSelector with:obj1 { ... }
- elementsPerform:(SEL)aSelector with:obj1 with:obj2 { ... }
- elementsPerform:(SEL)aSelector with:obj1 with:obj2 with:obj3 { ... }
```

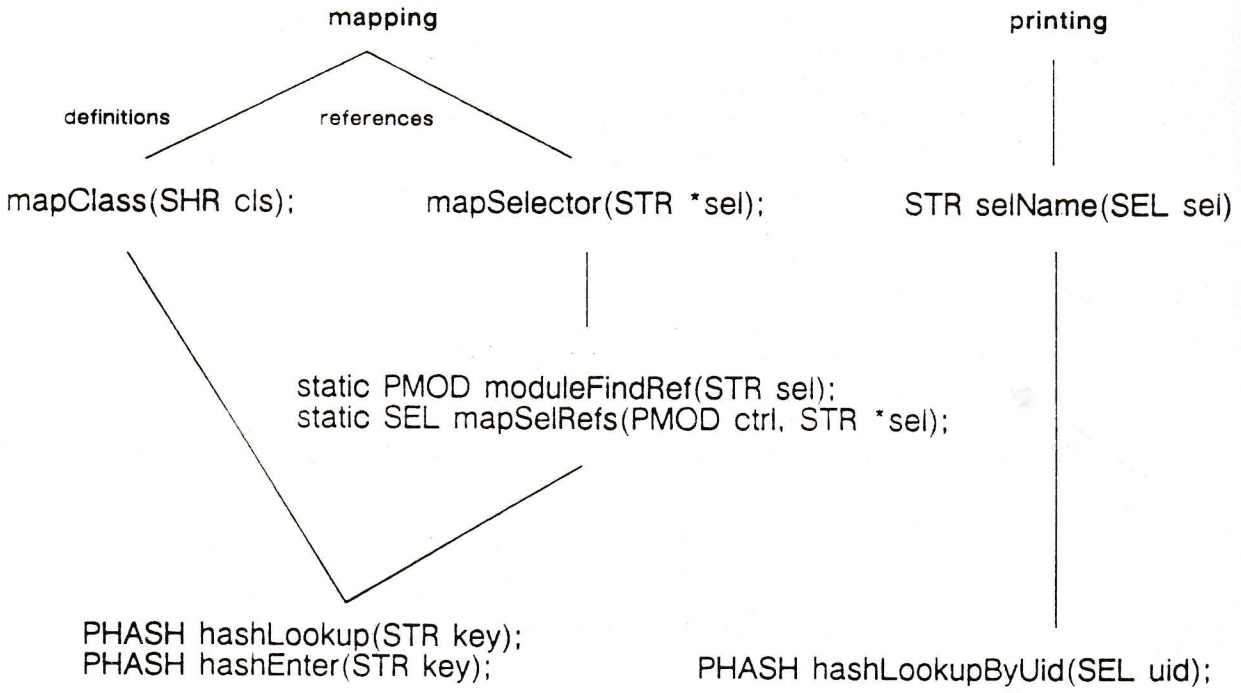
2. cvtToId(STR), __insertClass(SHR *)

3. _msg(id, SEL), _msgSuper(id, SEL), _msgImpFind(SHR, SEL, id *)

4. _prnFrame() { ... }, _osort(id *, int, SEL), __onExit(id, id, SEL)

5. "forloops.h" /* logic for looping through class/phyla tables */

New Selector facilities



typedef struct hashedSelector HASH, *PHASH;

PHASH	next
STR	key
SEL	uid

* the symbolic represention must now be obtained from **STR selName(SEL sel)** for printing.