Declared Properties

When the compiler encounters property declarations (see Declared Properties in *The Objective-C Programming Language*), it generates descriptive metadata that is associated with the enclosing class, category or protocol. You can access this metadata using functions that support looking up a property by name on a class or protocol, obtaining the type of a property as an `@encode` string, and copying a list of a property's attributes as an array of C strings. A list of declared properties is available for each class and protocol.

Property Type and Functions

The `Property` structure defines an opaque handle to a property descriptor.

```
typedef struct objc_property *Property;
```

You can use the functions `class_copyPropertyList` and `protocol_copyPropertyList` to retrieve an array of the properties associated with a class (including loaded categories) and a protocol respectively:

```
objc_property_t *class_copyPropertyList(Class cls, unsigned int *outCount)
objc_property_t *protocol_copyPropertyList(Protocol *proto, unsigned int *outCount)
```

For example, given the following class declaration:

```
@interface Lender : NSObject {
    float alone;
}
@property float alone;
@end
```

you can get the list of properties using:

```
id LenderClass = objc_getClass("Lender");
unsigned int outCount;
objc_property_t *properties = class_copyPropertyList(LenderClass, &outCount);
```

You can use the `property_getName` function to discover the name of a property:

```
const char *property_getName(objc_property_t property)
```

You can use the functions `class_getProperty` and `protocol_getProperty` to get a reference to a property with a given name in a class and protocol respectively:

```
objc_property_t class_getProperty(Class cls, const char *name)
objc_property_t protocol_getProperty(Protocol *proto, const char *name, BOOL
isRequiredProperty, BOOL isInstanceProperty)
```

You can use the `property_getAttributes` function to discover the name and the `@encode` type string of a property. For details of the encoding type strings, see Type Encodings; for details of this string, see Property Type String and Property Attribute Description Examples.

```
const char *property_getAttributes(objc_property_t property)
```

Putting these together, you can print a list of all the properties associated with a class using the following code:

```
id LenderClass = objc_getClass("Lender");
```

```
unsigned int outCount, i;
objc_property_t *properties = class_copyPropertyList(LenderClass, &outCount);
for (i = 0; i < outCount; i++) {
    objc_property_t property = properties[i];
    fprintf(stdout, "%s %s\n", property_getName(property),
property_getAttributes(property));
}
```

Property Type String

You can use the `property_getAttributes` function to discover the name, the `@encode` type string of a property, and other attributes of the property.

The string starts with a `T` followed by the `@encode` type and a comma, and finishes with a `V` followed by the name of the backing instance variable. Between these, the attributes are specified by the following descriptors, separated by commas:

Table 7-1 Declared property type encodings

Code	Meaning
`R`	The property is read-only (`readonly`).
`C`	The property is a copy of the value last assigned (`copy`).
`&`	The property is a reference to the value last assigned (`retain`).
`N`	The property is non-atomic (`nonatomic`).
`G<name>`	The property defines a custom getter selector name. The name follows the `G` (for example, `GcustomGetter,`).
`S<name>`	The property defines a custom setter selector name. The name follows the `S` (for example, `ScustomSetter:,`).
`D`	The property is dynamic (`@dynamic`).
`W`	The property is a weak reference (`__weak`).
`P`	The property is eligible for garbage collection.
`t<encoding>`	Specifies the type using old-style encoding.

For examples, see Property Attribute Description Examples.

Property Attribute Description Examples

Given these definitions:

```
enum FooManChu { FOO, MAN, CHU };
struct YorkshireTeaStruct { int pot; char lady; };
typedef struct YorkshireTeaStruct YorkshireTeaStructType;
union MoneyUnion { float alone; double down; };
```

the following table shows sample property declarations and the corresponding string returned by `property_getAttributes`:

Property declaration	Property description
`@property char charDefault;`	`Tc,VcharDefault`
`@property double doubleDefault;`	`Td,VdoubleDefault`
`@property enum FooManChu enumDefault;`	`Ti,VenumDefault`
`@property float floatDefault;`	`Tf,VfloatDefault`
`@property int intDefault;`	`Ti,VintDefault`
`@property long longDefault;`	`Tl,VlongDefault`
`@property short shortDefault;`	`Ts,VshortDefault`
`@property signed signedDefault;`	`Ti,VsignedDefault`
`@property struct YorkshireTeaStruct structDefault;`	`T{YorkshireTeaStruct="pot"i"lady"c},VstructDefault`
`@property YorkshireTeaStructType typedefDefault;`	`T{YorkshireTeaStruct="pot"i"lady"c},VtypedefDefault`
`@property union MoneyUnion unionDefault;`	`T(MoneyUnion="alone"f"down"d),VunionDefault`
`@property unsigned unsignedDefault;`	`TI,VunsignedDefault`
`@property int (*functionPointerDefault)(char *);`	`T^?,VfunctionPointerDefault`
`@property id idDefault;` Note: the compiler warns: `"no 'assign', 'retain', or 'copy' attribute is specified - 'assign' is assumed"`	`T@,VidDefault`
`@property int *intPointer;`	`T^i,VintPointer`
`@property void *voidPointerDefault;`	`T^v,VvoidPointerDefault`

`@property int intSynthEquals;`
In the implementation block:
`@synthesize intSynthEquals=_intSynthEquals;`
	`Ti,V_intSynthEquals`

`@property(getter=intGetFoo, setter=intSetFoo:) int intSetterGetter;`	`Ti,GintGetFoo,SintSetFoo:,VintSetterGetter`
`@property(readonly) int intReadonly;`	`Ti,R,VintReadonly`
`@property(getter=isIntReadOnlyGetter, readonly) int intReadonlyGetter;`	`Ti,R,GisIntReadOnlyGetter`
`@property(readwrite) int intReadwrite;`	`Ti,VintReadwrite`
`@property(assign) int intAssign;`	`Ti,VintAssign`
`@property(retain) id idRetain;`	`T@,&,VidRetain`
`@property(copy) id idCopy;`	`T@,C,VidCopy`
`@property(nonatomic) int intNonatomic;`	`Ti,VintNonatomic`

`@property(nonatomic, readonly, copy) id idReadonlyCopyNonatomic;`	`T@,R,C,VidReadonlyCopyNonatomic`
`@property(nonatomic, readonly, retain) id idReadonlyRetainNonatomic;`	`T@,R,&,VidReadonlyRetainNonatomic`