Protocol

Inherits From: Object

Declared In: objc/Protocol.h

Class Description

A Protocol object corresponds to a protocol declaration in the Objective€C language. It's the data structure that the run-time system uses to keep track of the protocol. Just as the compiler creates one class object for each class declaration it sees, it creates one Protocol object for each protocol declaration it encounters, provided the protocol is used somewhere within the program.

In Objective€C, protocols are declared with the **@protocol** directive:

```
@protocol Cartwheels
- turn:(int)numWheels startingFrom:(int)side;
- setRotationSpeed:(float)velocity;
- (BOOL)canStartFromRight;
- (BOOL)canStartFromLeft;
@end
```

The same directive, but with a set of trailing parentheses, is used to refer to a Protocol object in source code. In the following example, the Protocol object for the Cartwheels protocol is assigned to the **wheels** variable:

```
Protocol *wheels = @protocol(Cartwheels);
```

The **@protocol()** directive is the only way to ask for a Protocol object. The Protocol class doesn't define any methods that return or initialize instances of the class.

Because Protocol objects are built by the compiler, not by the application, and are part of the run-time system for the Objective€C language, they play a slightly different role within an application that most other objects. In particular, you should not allocate and initialize your own instances of the class. The only valid Protocol objects are those obtained through **@protocol()**.

Incorporation and Adoption

A protocol declaration can incorporate other protocols by listing them within angle brackets:

```
@protocol Tumbling <Cartwheels, WalkOvers, Flips, Aerials>
```

Class declarations use the same syntax to adopt protocols:

```
@interface Gymnast : Object <Tumbling, FloorRoutines>
```

Protocols can also be adopted in categories:

```
@interface Gymnast (BalanceBeam) <Dismounting>
```

The adopting class (or category) must implement all the methods declared in the protocol, including methods declared in any incorporated protocols. In the example above, the Gymnast class is obligated to implement all the methods declared in the Tumbling, Cartwheels, WalkOvers, Flips, Aerials, and FloorRoutines protocols; the BalanceBeam category of Gymnast must implement the methods declared in the Dismounting protocol. If any method is left undefined, the compiler will issue a warning.

You can ask a class if it adheres to a particular protocol by using the **conformsTo:** method defined in the Object class. This method returns YES if the receiving class, or any class above it in the inheritance hierarchy, directly or indirectly adopts the protocol. The same method can also be used to ask an instance if its class conforms:

```
if ( [myObject conformsTo:@protocol(Tumbling)] )
    [myObject turn:4 startingFrom:RIGHTSIDE];
```

Asking whether an object conforms to a protocol is very much like asking whether it responds to a messageÐexcept that **respondsTo:** tests whether one particular method is implemented and **conformsTo:** tests whether a group of methods has been adopted (and presumably implemented).

When sent to a Protocol object, a **conformsTo:** message asks if the receiver incorporates another protocol. The following message would return YES:

```
BOOL canFlip = [@protocol(Tumbling) conformsTo:@protocol(Flips)];
```

Type Checking

When a protocol name is included in a type specification, as in

```
id <Cartwheels, Flips> nadia;
```

or in

```
- setGymnast:(id <Tumbling>)anObject;
```

the compiler will check to make sure that only objects that conform to the specified protocols are used in those slots. Thus, protocols provide an added dimension of type checking at compile time.

Protocol Objects

The compiler creates a Protocol object for every protocol declared in source code, provided the protocol is also either:

- Adopted by a class, or
- Referred to by an **@protocol()** directive.

Simply using the protocol name in a type declaration isn't sufficient to cause a Protocol object to be created.

Instance Variables

None declared in this class.

Method Types

Getting the protocol name	- name
Testing for incorporated protocols	- conformsTo:
Getting method descriptions	- descriptionForInstanceMethod: - descriptionForClassMethod:

Instance Methods

conformsTo:

- (BOOL)**conformsTo:**(Protocol *)*aProtocol*

Returns YES if the receiving Protocol object directly or indirectly incorporates the *aProtocol* protocol, and NO if it doesn't. One protocol can incorporate another by declaring it within angle brackets:

```
@protocol BalanceBeam <Cartwheels, HandStands>
```

In the following code,

```
[@protocol(BalanceBeam) conformsTo:@protocol(Cartwheels)]
```

conformsTo: would return YES:

See also: + **conformsTo:** (Object)

descriptionForClassMethod:

- (struct objc_method_description *)**descriptionForClassMethod:**(SEL)*aSelector*

Returns a pointer to a structure describing the *aSelector* class method, or NULL if *aSelector* isn't declared as a class method in the receiving Protocol.

The structure has two fields, as illustrated below:

```
struct objc_method_description {
    SEL name;
    char *types;
};
```

The first field contains the method selector (which should be identical to *aSelector*). The second field contains encoded information about the method's return and argument types. Type information is encoded according to the conventions of the **@encode()** directive. For example, type information for this method

```
- (float)returnFloatForInt:(int)number
                 andString:(char *)name
                 andStruct:(struct entry)data;
```

would be encoded as:

```
f28@8:12i16*20{entry=**@}24
```

This method returns a **float** (`f') and pushes 28 bytes onto the stack. Its first two arguments are an object (`@') at an offset of 8 bytes from the stack pointer and a selector (`:') at an offset of 12 bytes. These two arguments correspond to **self** (the message receiver) and **_cmd** (the method selector), which are present in every method implementation but are normally hidden by the Objective€C language. The three declared arguments are an **int** (`i') at an offset of 16 bytes, a string (`*') at an offset of 20 bytes, and a structure (ª{...}º) at an offset of 24 bytes. The structure name is ªentryº and it consists of two character pointers and an object **id** (ª**@º).

See also: - **descriptionForInstanceMethod:**, - **descriptionForMethod:** (Object)

descriptionForInstanceMethod:

- (struct objc_method_description *)
  descriptionForInstanceMethod:(SEL)*aSelector*

Returns a pointer to a structure describing the *aSelector* instance method, or NULL if the *aSelector* method isn't declared as an instance method in the receiving Protocol. The structure is described under **descriptionForClassMethod:** above.

See also: - **descriptionForClassMethod:**, - **descriptionForMethod:** (Object)

name

- (const char *)**name**

Returns a null-terminated string containing the name of the protocol.