;D_CurrencyConverter_DefineClasses.rtfd;;¬ Previous Section ;F_CurrencyConverter_Build.rtfd;;¬ Next Section

2. Currency Converter Tutorial

Implementing the Classes of Currency Converter

Interface Builder generates source code files from the (partial) class definitions you've made. These files are ªskeletal,º in the sense that they contain little more than essential Objective C directives and the class-definition information. You'll usually need to supplement these files with your own code.

1 In Interface Builder, generate header and implementation files.

Go to the Classes display of the nib file window.

Select the ConverterController class.

Choose Create Files from the Operations pull-down menu.

_IB_CreateFilesA.eps ¬

Interface Builder then displays two attention panels, one after the other:

When a Create Files panel is displayed, click Yes.

A second Create Files panel is displayed; click Yes again.

Repeat for the Converter class.

Save the nib file.

_IB_CreateFilesB.eps ¬

Now we leave Interface Builder for this application. You'll complete the application using Project Builder.

2 Examine an interface (header) file in Project Builder.

Hide Interface Builder and activate Project Builder.

Click Headers in the project browser.

Select **ConverterController.h**.

_PB_TemplateHeader.eps ¬

You can add instance variables or method declarations to a header file generated by Interface Builder. This is commonly done, but it isn't necessary in ConverterController's case. But we do need to add a method to the Converter class that the ConverterController object can invoke to get the result of the computation. Let's start with by declaring the method in **Converter.h**.

3 Add a method declaration.

Select **Converter.h** in the project browser.

Insert a declaration for **convertAmount:byRate:**.

```
#import <AppKit/AppKit.h>
#import <Foundation/Foundation.h>

@interface Converter:NSObject
{
}
- (float)convertAmount:(float)rate byRate:(float)amt;

@end
```

This declaration states that **convertAmount:byRate:** takes two arguments of type **float**, and returns a **float** value. When parts of a method name have colons, such as **convertAmount:** and **byRate:**, they are *keywords* which introduce arguments. (These are keywords in a sense different from keywords in the C language.) Most

method declarations begin with a dash (-), followed by a space.

Now you need to update both implementation files. First examine **Converter.m**.

4 Examine an implementation file.

Click Classes in the project browser.
Select **Converter.m**.

_PB_BlankMFile.eps ¬

For this class, implement the method declared in **Converter.h**. Between **@implementation Converter** and **@end** add the following code:

5 Implement the classes.

Type the code below between **@implementation** and **@end** in **Converter.m**.

```
- (float)convertAmount:(float)amt byRate:(float)rate
{
    return (amt * rate);
}
```

The method simply multiplies the two arguments and returns the result. Simple enough. Next update the ªemptyº implementation of the **convert:** method that Interface Builder generated.

Select **ConverterController.m** in the project browser.
Update the **convert:** method as shown in the example below.
Import **Converter.h**.

```
- (void)convert:(id)sender
{
    float rate, amt, total;
```

```
        amt = [dollarField floatValue]; €€€€€€€€€€€€€€€€€€€€/* 1 */
        rate = [rateField floatValue];
        total = [converter convertAmount:amt byRate:rate];  /* 2 */
        [totalField setFloatValue:total]; €€€€€€€€€€€€€€€€€€/* 3 */
        [rateField selectText:self]; €€€€€€€€€€€€€€€€€€€€€€/* 4 */
    }
```

The **convert:** method does the following:

1. Gets the floating-point values typed into the rate and dollar-amount fields
2. Invokes the **convertAmount:byRate:** method and gets the returned value.
3. Uses **setFloatValue:** to write the returned value in the Amount in Other Currency text field (**totalField**).
4. Sends **selectText:** to the rate field; this puts the cursor in the rate field so the user begin another calculation.

Be sure to **#import ªConverter.hº**ÐConverterController invokes a method defined in the Converter class, so it needs to be aware of the method's declaration.

Related Concept: ;CurrencyConverterConcepts.rtfd;linkMarkername ObjectiveCQuickReference;, Objective-C Quick Reference

144635_TableRule.eps ¬Before You Go On

Each line of the **convert:** method shown above, excluding the declaration of **float**s, is a message. The ªwordº on the left side of a message expression identifies the object receiving the message (called the ªreceiverº). These objects are identified by the outlets you defined and connected. After the receiver comes the name of the method that the sending object (called the ªsenderº) wants to invoke. Messages often return values; in the above example, the local variables **rate**, **amt**, and **total** hold these values.

435874_TableRule.eps ¬
Before you build the project, add a small bit of code to **ConverterController.m** that will make life a little easier for your users. When the application starts up, you want Currency Converter's window to be selected and the cursor to be in the Exchange Rate per $1 field. We can do this only after the nib file is unarchived, which establishes the connection to the text field **rateField**. To enable set-up operations like this, Interface Builder sends **awakeFromNib** to all objects when it finishes unarchiving. Implement this method to take appropriate action.

6 Implement the awakeFromNib method to perform start-up tasks.

```
- (void)awakeFromNib
{
    [rateField selectText:self]; /* 1 */
    [[rateField window] makeKeyAndOrderFront:self]; /* 2 */
}
```

1. You've seen the **selectText:** message before, in the **convert:** implementation; it selects the text in the text field that receives the message, inserting the cursor if there is no text.

2. The **makeKeyAndOrderFront:** message does as it says: It makes the receiving window the key window and puts it before all other windows on the screen. This message also *nests* another message; **[rateField window]** returns the window to which the text field belongs, and the **makeKeyAndOrderFront:** method is then sent to this returned object.