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

2. Currency Converter Tutorial

Designing the Currency Converter Application

An object-oriented application should be based on a design that identifies the objects of the application and clearly defines their roles and responsibilities. You normally work on a design before you write a line of code. You don't need any fancy tools for designing many applications; a pencil and a pad of paper will do.

Currency Converter is an extremely simple application, but there's still a design behind it. This design is based upon the Model-View-Controller paradigm, a model behind many designs for object-oriented programs (see ``The Model-View-Controller Paradigm'' on page 33). This design paradigm aids in the development of maintainable, extensible, and understandable systems. But first, you might want to read the sidebar below to understand the symbol used in the design diagram.

> **Note:** This design for Currency Converter is intended to illustrate a few points, and so is perhaps overly designed for something so simple. It is quite possible to have the application's controller class, ConverterController, do the computation and do without the Converter class..

You can divide responsibility within Currency Converter among two custom objects and the user interface, taken as a collection of ready-made Application Kit objects. The Converter object is responsible for computing a currency amount and returning that value. Between the user interface and the Converter object is a *controller object*, ConverterController. ConverterController coordinates the activity between the Converter object and the UI objects.

CC_Design.eps ¬

The ConverterController class assumes a central role. Like all controller objects, it communicates with the interface and with model objects, and it handles tasks specific to the application, such as managing the cursor. ConverterController gets the values users enter into fields, passes these values to the Converter object, gets

the result back from Converter, and puts this result in a field in the interface.

The Converter class merely computes a value from two arguments passed into it and returns the result. As with any model object, it could also hold data as well as provide computational services. Thus, objects that represent customer records (for example) are akin to Converter. By insulating the Converter class from application-specific details, the design for Currency Converter makes it more reusable, as you'll see in the Travel Advisor tutorial.

Related Concepts: ;CurrencyConverterConcepts.rtfd;linkMarkername WhyanObjectisLikeaJellyDonut;, Why an Object is Like a Jelly Donut

;CurrencyConverterConcepts.rtfd;linkMarkername TheModel-View-ControllerParadigm;, The Model-View-Controller Paradigm