;D_TravelAdvisor_ImplCountry.rtfd;;¬ Previous Section

3. Travel Advisor Tutorial

Implementing the TAController Class

The TAController class plays a central role in the Travel Advisor application. As the application's controller object, it transfers data from the model objects (Country instances) to the fields of the interface and, when users enter or modify data, back to the correct Country object. The TAController must also coordinate the data displayed in the table view with the current object, and it must do the right thing when users select an item in the table view or click the Add or Delete button. All custom code specific to the user interface resides in TAController.

The mechanics of this activity require an array (NSMutableArray) and a dictionary (NSMutableDictionary) for storing and accessing Country data. The following diagram illustrates the relationship among interface components, TAController, and the sources of data.

TA_DataDesign.eps ¬

The dictionary contains Country objects (values) that are identified by the names of countries (keys). The dictionary is the source of data for the fields of Travel Advisor. The array derives from the dictionary and is sorted. It is the source of data for the table view.

After describing what other instance variables you must add to TAController, this section covers the following implementation tasks:

425833_SquareBullet.eps ¬	Getting the data from Country objects to the interface and back
695106_SquareBullet.eps ¬	Getting the table view to work, including updating Country records

280092_SquareBullet.eps ¬ Adding and deleting ªrecordsº (Country objects)
621271_SquareBullet.eps ¬ Formatting and validating field values
443190_SquareBullet.eps ¬ ªHousekeepingº tasks (application management)

1 Update TAController.h.

Import **Country.h.**

Add the instance-variable declarations shown below.

```
NSMutableDictionary *countryDict;
NSMutableArray      *countryKeys;
BOOL                recordNeedsSaving;
```

The variables **countryDict** and **countryKeys** identify the array and the dictionary discussed on the previous page. The boolean **recordNeedsSaving** flags that record if the user modifies the information in any field.

Add the enum declaration shown below between the last **#import** directive and the **@interface** directive.

```
enum LogisticsFormTags {
    LGairports=0,
    LGairlines,
    LGtransportation,
    LGhotels
};
```

This declaration is not essential, but the **enum** constants provide a clear and convenient way to identify the cells in the Logistics form. Methods such as **cellAtIndex:** identify the editable cells in a form through zero-based indexing. This declaration gives each cell in the Logistics form a meaningful designation.

Related Concept: ;TravelAdvisorConcepts.rtfd;linkMarkername TurboCodingWithProjectBuilder;, Turbo Coding With Project Builder

DataMediation;¬Data Mediation

TAController acts as the mediator of data exchanged between a source of data and the display of that data. Data mediation involves taking data from fields, storing it somewhere, and putting it back into the fields later. TAController has two methods related to data mediation: **populateFields:** puts Country instance data into the fields of Travel Advisor and **extractFields:** updates a Country object with the information in the fields.

2 Implement the methods that transfer data to and from the application's fields.

Implement the **populateFields:** method as below.

```
- (void)populateFields:(Country *)aRec
{
    [countryField setStringValue:[aRec name]];          /* 1 */

    [[logisticsForm cellAtIndex:LGairports] setStringValue:
        [aRec airports]];                               /* 2 */
    [[logisticsForm cellAtIndex:LGairlines] setStringValue:
        [aRec airlines]];
    [[logisticsForm cellAtIndex:LGtransportation] setStringValue:
        [aRec transportation]];
    [[logisticsForm cellAtIndex:LGhotels] setStringValue:
        [aRec hotels]];

    [currencyNameField setStringValue:[aRec currencyName]];
    [currencyRateField setFloatValue:[aRec currencyRate]];
    [languagesField setStringValue:[aRec languages]];
```

```
    [englishSpokenSwitch setState:[aRec englishSpoken]];

    [commentsField setString:[aRec comments]];

    [countryField selectText:self];                    /* 3 */
}
```

1. Causes the Country field to display the value of the **name** instance variable of the Country record (**aRec**) passed into the method. Since **[aRec name]** is nested, the object it returns is used as the argument of **setStringValue:**, which sets the textual content of the receiver (in this case, an NSTextFieldCell).
2. The **cellAtIndex:** message is sent to the form and returns the cell identified by the **enum** constant LGairports.
3. Selects the text in the Country field or, if there is no text, inserts the cursor.

Although it doesn't do anything with data, the **blankFields:** method is similar in structure to **populateFields:**. The **blankFields:** method clears whatever appears in Travel Advisor's fields by inserting empty string objects and zeros.

Implement the **blankFields:** method as shown below.

```
- (void)blankFields:(id)sender
{
    [countryField setStringValue:@""];

    [[logisticsForm cellAtIndex:LGairports] setStringValue:@""];
    [[logisticsForm cellAtIndex:LGairlines] setStringValue:@""];
    [[logisticsForm cellAtIndex:LGtransportation] setStringValue:@""];
```

```
    [[logisticsForm cellAtIndex:LGhotels] setStringValue:@""];

    [currencyNameField setStringValue:@""];
    [currencyRateField setFloatValue:0.000];
    [languagesField setStringValue:@""];
    [englishSpokenSwitch setState:NO];                  /* 1 */

    [currencyDollarsField setFloatValue:0.00];
    [currencyLocalField setFloatValue:0.00];
    [celsius setIntValue:0];

    [commentsField setString:@"" ];                     /* 2 */
    [countryField selectText:self];
}
```

1. The **setState:** message affects the appearance of two-state toggled controls, such as a switch button. With an argument of YES, the checkmark appears; with an argument of NO, the checkmark is removed.
2. The **setString:** message sets the textual contents of NSText objects (such as the one enclosed by the scroll view).

798729_TableRule.eps ¬***Before You Go On***

Exercise: Implement the **extractFields:** method. In this method set the values of the passed-in Country record's instance variables with the contents of the associated fields.

Tip: Use the **stringValue** method to get field contents and use Country's accessor methods to set the values of instance variables.
279392_TableRule.eps ¬**Related Concept:** ;TravelAdvisorConcepts.rtfd;linkMarkername

FindingInformationWithinYourProject;, Finding Information Within Your Project

GettingtheTableViewtoWork;¬Getting the Table View to Work

Table views are objects that display data as records (rows) with attributes (columns). The table view in Travel Advisor displays the simplest kind of record, with each record having only one attribute: a country name.

Table views get the data they display from a *data source*. A data source is an object that implements the informal NSTableDataSource protocol to respond to NSTableView requests for data. Since the NSTableView organizes records by zero-based indexing, it is essential that the data source organizes the data it provides to the NSTableView similarly: in an array.

3 Implement the behavior of the table view's data source.

In TAController's **awakeFromNib** method, create and sort the array of country names.

In the same method, designate **self** as the data source.

```
- (void)awakeFromNib
{
    NSArray *tmpArray = [[countryDict allKeys]          /* 1 */
        sortedArrayUsingSelector:@selector(compare:)];
    countryKeys = [[NSMutableArray alloc] initWithArray:tmpArray];

    [tableView setDataSource:self];                     /* 2 */
    [[[tableView tableColumns] objectAtIndex:0]         /* 3 */
        setIdentifier:@"Countries"];
    [tableView sizeLastColumnToFit];
}
```

1. The **[countryDict allKeys]** message returns an array of keys (country names) from the unarchived dictionary that contains Country objects as values. The **sortedArrayUsingSelector:** message sorts the items in this ªrawº array using the **compare:** method defined by the class of the objects in the array, in this case NSString (this is an example of polymorphism and dynamic binding). The sorted names go into a temporary NSArrayÐsince that is the type of the returned valueÐand this temporary array is used to create a mutable array, which is then assigned to **countryKeys**. A mutable array is necessary because users may add or delete countries from the application.

2. The **[tableView setDataSource:self]** message identifies the TAController object as the table view's data source. The table view will commence sending NSTableDataSource messages to TAController. (You can effect the same thing by setting the NSTableView's **dataSource** outlet in Interface Builder.)

If users are supposed to edit the cells of the table view, you would also make TAController the delegate of the table view at this point (with **setDelegate:**).The delegate receives messages relating to the editing and validation of cell contents. For details, see the specification on NSTableView in the Application Kit reference documentation.

3. Every column has an *identifier* to associate it with a column, which is itself usually associated with an attribute. By default, the identifier is a number: the first column is 0, the second column is 1, and so on. This compound message makes the identifier a string object and thus binds it semantically to the attribute. The **tableColumns** method returns all NSTableColumns in a array; in this case, only the single column of this table view. The **setIdentifier:** message sets the value.

To fulfill its role as data source, TAController must implement two methods of the NSTableDataSource informal protocol.

Implement two methods of the NSTableDataSource informal protocol:

-€**numberOfRowsInTableView:**

€€ -€**tableView:objectValueForTableColumn:row:**

```
- (int)numberOfRowsInTableView:(NSTableView *)theTableView
{                                                          /* 1 */
    return [countryKeys count];
}

- (id)tableView:(NSTableView *)theTableView                /* 2 */
      objectValueForTableColumn:(NSTableColumn *)theColumn
            row:(int)rowIndex
{
    if ([[theColumn identifier] isEqualToString:@"Countries"])
        return [countryKeys objectAtIndex:rowIndex];
    else
        return nil;
}
```

1. Returns the number of country names in the **countryKeys** array.

   If you had an application with multiple table views, each would invoke this NSTableView delegation method (as well as the others). By evaluating the **theTableView** argument, you could distinguish which table view was involved.

2. This method first evaluates the column identifier to determine if it's the right column (it *should* always return ªCountriesº). If it is, the method returns the country name from the **countryKeys** array that is associated with **rowIndex**. This name is then displayed at **rowIndex** of the column. (Remember, the array and the cells of the column are synchronized in terms of their indexing.)

The NSTableDataSource informal protocol has a another method, **tableView:setObjectValue:forTableColumn:row:**, that you won't implement in this tutorial. This method allows the data source to extract data entered by users into table-view cells; since Travel Advisor's table view is read-only, there is no need to implement it.

Related Concept: ;TravelAdvisorConcepts.rtfd;linkMarkername GettinginontheAction:DelegationandNotification;, Getting in on the Action: Delegation and Notification

The final thing you need to do to get the table view working is to respond to mouse clicks in it. As you recall, you defined in Interface Builder the **handleTVClick:** action for this purpose. This method must do a number of things:

SquareBullet.eps ¬ Save the current Country object or create a new one.
635409_SquareBullet.eps ¬ If there's a new record, re-sort the array providing data to the table view.
753752_SquareBullet.eps ¬ Display the selected record.

4 Update records.

Implement the method that responds to user selections in the table view.

```
- (void)handleTVClick:(id)sender
{
    Country *aRec, *newRec, *newerRec;
    int index;

    /* does current obj need to be saved? */
    if (recordNeedsSaving) {                                    /* 1 */
        /* is current object already in dictionary? */
        if (aRec=[countryDict objectForKey:[countryField stringValue]]) {
            /* remove if it's been changed */
            if (aRec) {
                NSString *country = [aRec name];
                [countryDict removeObjectForKey:country];
```

```
                [countryKeys removeObject:country];
            }
        }
        /* Create Country obj, add to dict, add name to keys array */
        newRec = [[Country alloc] init];
        [self extractFields:newRec];
        [countryDict setObject:newRec forKey:[countryField stringValue]];
        [countryKeys addObject:[countryField stringValue]];

        /* sort array here */
        [countryKeys sortUsingSelector:@selector(compare:)];
        [tableView tile];
    }
    index = [sender selectedRow];
    if (index >= 0 && index < [countryKeys count]) {          /* 2 */
        newerRec = [countryDict objectForKey:
            [countryKeys objectAtIndex:index]];
        [self populateFields:newerRec];
        [commentsLabel setStringValue:[NSString stringWithFormat:
            @"Notes and Itinerary for %@", [countryField stringValue]]];
        recordNeedsSaving=NO;
    }
}
```

This method has two major sections, each introduced by an **if** statement.

1. When any Country-object data is added or altered, Travel Advisor sets the **recordNeedsSaving** flag to YES (you'll learn how to do this on later on). If **recordNeedsSaving** is YES, first delete any existing Country record for that country from the dictionary and also remove the country name from the table

view's array. (Upon removal, the objects are automatically released by the array.) Then create a new Country instance and initialize it with the values currently on the screen; add the instance to the dictionary, add the country name to the table view's array, sort the array, and reset the **recordNeedsSaving** flag. At the end, invoke the **tile** method , which (among other things) causes the table view to request data from its data source.

2. The **[sender selectedRow]** message queries the table view for the row index of the cell that was clicked. If this index is within expected bounds, use it to get the country name from the array, and then use the country name as the key to get the associated Country instance. Write the instance-variable values of this instance to the fields of the application, update the ªNotes and Itinerary forº label.

900826_TableRule.eps ¬Optional Exercise

Application developers often like to have key alternatives to mouse actions such as clicking a table view. One way of acquiring a key alternative is to add a menu cell in Interface Builder, specify a key as an attribute of the cell, define an action method that will be invoked, and then implement that method.

The methods **nextRecord:** and **prevRecord:** should be invoked when users chose Next Record and Prev Record or type the key equivalents Command-n and Command-r. In **TAController.m**, implement these methods, keeping the following hints in mind:

1. Get the index of the selected row (**selectedRow**).

2. Increment or decrement this index, according to which key is pressed (or which command is clicked).

3. If the start or end of the table view is encountered, ªwrapº the selection.
   (Hint: Use the index of the last object in the **countryKeys** array.)

4. Using the index, select the new row, but don't extend the selection.

5. Simulate a mouse click on the new row by sending **handleTVClick:** to **self**.

115903_TableRule.eps ¬

AddingandDeletingRecords;¬Adding and Deleting Records

When users click Add Record to enter a Country ªrecord,º the **addRecord:** method is invoked. You want this method to do a few things besides adding a Country object to the application's dictionary:

327321_SquareBullet.eps ¬ Ensure that a country name has been entered.
435549_SquareBullet.eps ¬ Make the table view reflect the new record.
543188_SquareBullet.eps ¬ If the record already exists, update it (but only if it's been modified).

5 Implement the method that adds a Country object to the NSDictionary ªdatabase.º

```
- (void)addRecord:(id)sender
{
    Country *aCountry;
    NSString *countryName = [countryField stringValue];
/* 1 */
   if (countryName && (![countryName isEqualToString:@""])) {
        aCountry = [countryDict objectForKey:countryName];
        if (aCountry && recordNeedsSaving) {
           /* remove old Country object from dictionary */
           [countryDict removeObjectForKey:countryName];
           [countryKeys removeObject:countryName];
           aCountry = nil;
         }
        if (!aCountry)  /* record is new or has been removed */
            aCountry = [[Country alloc] init];
```

```
        else /* record already exists and hasn't changed */
            return;
/* 2 */
        [self extractFields:aCountry];
        [countryDict setObject:aCountry forKey:[aCountry name]];
        [countryKeys addObject:[aCountry name]];
        [countryKeys sortUsingSelector:@selector(compare:)];
/* 3 */
        recordNeedsSaving=NO;
        [commentsLabel setStringValue:[NSString stringWithFormat:
            @"Notes and Itinerary for %@",[countryField stringValue]]];
        [countryField selectText:self];
/* 4 */
        [tableView tile];
        [tableView selectRow:[countryKeys indexOfObject:
            [aCountry name]] byExtendingSelection:NO];
    }
}
```

1. This section of code verifies that a country name has been entered and sees if there is a Country object in the dictionary. If there's no object for the key, **objectForKey:** returns **nil**. If the object exists and it's flagged as modified, the code removes it from the dictionary and removes the country name from the **countryKeys** array. Note that removing an object from a dictionary or array also releases it, so the code sets **aCountry** to **nil**. It then tests **aCountry** and, if it's **nil**, creates a new object; otherwise it just returns, because an object already exists for this country and it hasn't been modified.

2. After updating the new Country object with the information on the application's fields (**extractFields:**), this code adds the Country object to the dictionary and the country name to the **countryKeys** array.

3. This section of code performs some things that have to be done, such as resetting the **recordNeedsSaving** flag and updating the label over the scroll view to reflect the just-added country.

4. The **tile** message forces the table view to update its contents. The **selectRow:byExtendingSelection:** message highlights the new record in the table view.

680453_TableRule.eps ¬Before You Go On

Exercise: Implement the **deleteRecord:** method. Although similar in structure to **addRecord:** this method is much simpler, because you don't need to worry about whether a Country record has been modified. Once you've deleted the record, remember to update the table view and clear the fields of the application.
986009_TableRule.eps ¬
Related Concept: ;TravelAdvisorConcepts.rtfd;linkMarkername AbstractClassesandClassClusters;, Abstract Classes and Class Clusters

FieldFormattingandValidation;¬Field Formatting and Validation

Travel Advisor has several numeric fields. Some display temperatures while others display currency amounts. In this stage, you'll enable these fields to format their contents by using a formatting API defined in the Application Kit.

6 Format and validate numeric fields.

Set the entry type and floating-point format of some TAController fields in the **awakeFromNib** method.

```
- (void)awakeFromNib
{
    [[currencyRateField cell] setEntryType:NSFloatType];
    [[currencyRateField cell] setFloatingPointFormat:YES
```

```
        left:2 right:1];
    [[currencyDollarsField cell] setEntryType:NSFloatType];
    [[currencyDollarsField cell] setFloatingPointFormat:YES left:5
        right:2];
    [[currencyLocalField cell] setEntryType:NSFloatType];
    [[currencyLocalField cell] setFloatingPointFormat:YES left:5
        right:2];
    [[celsius cell] setEntryType:NSFloatType];
    [[celsius cell] setFloatingPointFormat:YES left:2 right:1];
}
```

The NSCell class provides methods for specifying how cell values are formatted. In this instance, **setEntryType:** sets the type of value as a **float** and **setFloatingPointFormat:left:right:** specifies the number of digits on each side of the decimal point.

The NSControl class gives you an API for validating the contents of cells. Validation verifies that the values of cells fall within certain limits or meet certain criteria. In Travel Advisor, we want to make sure that the user does not enter a negative value in the Rate field.

The request for validation is a messageÐ**control:isValidObject:**Ðthat a control sends to its delegate. The control, in this case, is the Rate field.

In **awakeFromNib**, make TAController a delegate of the field to be validated.

```
[currencyRateField setDelegate:self];
```

Implement the **control:isValidObject:** method to validate the value of the field.

```
- (BOOL)control:(NSControl *)control isValidObject:(id)obj
{
```

```
    if (control == currencyRateField) {             /* 1 */
        if ([obj floatValue] < 0.0) {
           NSRunAlertPanel(@"Travel Advisor",       /* 2 */
            €€€€ @"Rate cannot be negative.", nil, nil, nil);
            return NO;
        }
    }
    return YES;
}
```

1. Because you might have more than one field's value to validate, this example first determines which field is sending the message. It then checks the field's value (passed in as the second object); if it is negative, it displays an attention panel and returns NO, blocking the entry of the value. Otherwise, it returns YES and the field accepts the value.

2. The **NSRunAlertPanel()** function allows you to display a modal attention panel from any point in your code. The above example calls this function simply to inform the user why the value cannot be accepted.

   _NSRunAlertExample.eps ¬

   Although Travel Advisor doesn't evaluate it, the **NSRunAlertPanel()** function returns a constant indicating which button the user clicks on the panel. The logic of your code could therefore branch according to user input. In addition, the function allows you to insert variable information (using **printf()**-style conversion specifiers) into the body of the message.

   For more information on **NSRunAlertPanel()**, see the ªFunctionsº section of the Application Kit (framework) reference documentation.

Related Concept: ;TravelAdvisorConcepts.rtfd;linkMarkername
Behindª ClickHereº:Controls,Cells,andFormatters;, Behind ª Click Hereº: Controls, Cells, and Formatters

ApplicationManagement;¬Application Management

By now you've finished the major coding tasks for Travel Advisor. All that remains to implement are a half dozen or so methods. Some of these methods perform tasks that every application should do. Others provide bits of functionality that Travel Advisor requires. In this section you'll:

411832_SquareBullet.eps ¬	Archive and unarchive the TAController object.
523633_SquareBullet.eps ¬	Implement TAController's **init** and **dealloc** methods.
649260_SquareBullet.eps ¬	Save data when the application terminates.
768828_SquareBullet.eps ¬	Mark the current record when users make a change.
891504_SquareBullet.eps ¬	Obtain and display converted currency values.

The data that users enter into Travel Advisor should be saved in the file system, or *archived*. The best time to initiate archiving in Travel Advisor is when the application is about to terminate. Earlier you made TAController the delegate of the application object (NSApp). Now respond to the delegate message **applicationShouldTerminate:**, which is sent just before the application terminates.

7 Archive the application's objects when it terminates.

Implement the delegate method **applicationShouldTerminate:**, as shown below.

```
- (BOOL)applicationShouldTerminate:(id)sender
{                                                          /* 1 */
    NSString *storePath = [[[NSBundle mainBundle] bundlePath]
        stringByAppendingPathComponent:@"TravelData"];
   /* save current record if it is new or changed */
```

```
    [self addRecord:self];
                                                              /* 2 */
    if (countryDict && [countryDict count])
         [NSArchiver archiveRootObject:countryDict toFile:storePath];

     return YES;
}
```

1. Constructs a pathname to the application wrapper in which to store the archive file ªTravelData.º The application wrapperÐthe ªhiddenº directory holding the application executable and required resourcesÐis a bundle, so NSBundle methods are used to get the bundle path.

2. If the **countryDict** dictionary holds Country objects, TAController archives it with the NSArchiver class method **archiveRootObject:toFile:**. Since the dictionary is designated as the root object for archiving, all objects that the dictionary references (that is, the Country objects it contains) will be archived too.

8 Implement TAController's methods for initializing and deallocating itself.

Implement the **init** method, as shown below.

Implement the **dealloc** method to release object instance variables.

```
- (id)init
/* 1 */
    NSString *storePath = [[NSBundle mainBundle]
        pathForResource:@"TravelData" ofType:nil];
    [super init];
/* 2 */
    countryDict = [NSUnarchiver unarchiveObjectWithFile:storePath];
/* 3 */
    if (!countryDict) {
```

```
        countryDict = [[NSMutableDictionary alloc] init];
        countryKeys = [[NSMutableArray alloc] initWithCapacity:10];
    } else
        countryDict = [countryDict retain];
    recordNeedsSaving=NO;

    return self;
}
```

1. Using NSBundle methods, locates the archive file ªTravelDataº in the application wrapper and returns the path to it.
2. The **unarchiveObjectWithFile:** message *unarchives* (that is, restores) the object whose attributes are encoded in the specified file. The object that is unarchived and returned is the NSDictionary of Country objects (**countryDict**).
3. If no NSDictionary is unarchived, the **countryDict** instance variable remains **nil**. If this is the case, TAController creates an empty **countryDict** dictionary and an empty **countryKeys** array. Otherwise, it retains the instance variable.

When users modify data in fields of Travel Advisor, you want to mark the current record as modified so later you'll know to save it. The Application Kit broadcasts a notification whenever text in the application is altered. To receive this notification, add TAController to the list of the notification's observers.

Related Concept: ;TravelAdvisorConcepts.rtfd;linkMarkername FlatteningtheObjectNetwork:CodingandArchiving;, Flattening the Object Network: Coding and Archiving

9 Write the code that marks records as modified.

In the **awakeFromNib** method, make TAController an observer of NSControlTextDidChangeNotification.

```
[[NSNotificationCenter defaultCenter] addObserver:self
     selector:@selector(textDidChange:)
     name:NSControlTextDidChangeNotification object:nil];
```

Next, implement the method that you indicated would respond to the notification; this method sets a flag, thereby marking the record as changed.

Implement **textDidChange:** to set the recordNeedsSaving flag.

```
- (void)textDidChange:(NSNotification *)notification
{
    if ([notification object] == currencyDollarsField ||
        [notification object] == celsius) return;

    recordNeedsSaving=YES;
}
```

Two of the editable fields of Travel Advisor hold temporary values used in conversions and so are not saved. This statement checks if these fields are the ones originating the notification and, if they are, returns without setting the flag. (The **object** message obtains the object associated with the notification.)

> You post notifications and add objects as observers of notifications with methods defined in the NSNotificationCenter class. NSNotification defines methods for creating notification objects and for accessing their attributes. For more information on these classes, see their specifications in the Foundation framework reference documentation.

The final method to implement is almost identical to the one you wrote for Currency Converter to display the results of a currency conversion when the user clicks the Convert button for currency conversion.

10 Implement the method that responds to a request for a currency conversion.

```
- (void)convertCurrency:(id)sender
{
    [currencyLocalField setFloatValue:
        [converter convertAmount:[currencyDollarsField floatValue]
        byRate:[currencyRateField floatValue]]];
}
```

628383_TableRule.eps ¬***Optional Exercise***

Convert Celsius to Fahrenheit: Implement the **convertCelsius:** method. You've already specified and connected the necessary outlets (**celsius**, **fahrenheit**) and action (**convertCelsius:**), so all that remains is the method implementation. The formula you'll need is:

F° = 9/5C° + 32

278927_TableRule.eps ¬

Building and Running Travel Advisor

When Travel Advisor is built, start it up by double-clicking the icon in the File Manager. Then put the application through the following tests:

785715_SquareBullet.eps ¬ Enter a few records. Make up geographical information if you have toÐ you're not trusting your future travels to this application. Not yet, anyway.

22461_SquareBullet.eps ¬ Click the items in the table view and notice how the selected records are displayed. Press Command-n and Command-r and observe what happens.

130912_SquareBullet.eps ¬ Enter values in the conversion fields to see how they're automatically

formatted. Try to enter a negative value in the Rate field.

239711_SquareBullet.eps ¬ Quit the application and then start it up again. Notice how the application displays the same records that you entered.

Related Concepts: ;TravelAdvisorConcepts.rtfd;linkMarkername UsingtheGraphicalDebugger;, Using the Graphical Debugger

;TravelAdvisorConcepts.rtfd;linkMarkername TipsforEliminatingDocumentationBugs;¬ Tips for Eliminating Deallocation Bugs