CHAPTER 5

Repetition Blocks, Times, and Dates

Topics

- The `Notifier` Component
- The `while` Loop
- The `for each` Loop
- The `Clock` Component
- The `DatePicker` Component

The Notifier Component

- The `Notifier` is a nonvisible component that allows an app to display dialog boxes.
- The `Notifier` Component displays the following boxes.

Figure 5-1 Dialog Boxes Displayed by the Notifier Component (*Source:* MIT App Inventor 2)

The Notifier Component

Message dialog

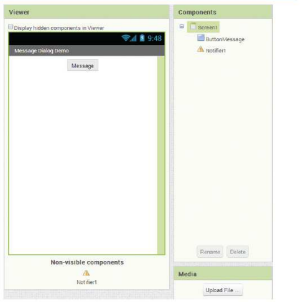
- A window that displays a title and a message.
- Waits for the user to click a button.

The `Notifier` Component

The Message Dialog

- Display a message dialog by calling the `Notifier` component's `ShowMessageDialog` method.
- The Project has a button named `ButtonMessage` and a `Notifier` named `Notifier1`.
- User clicks the button and a message dialog appears.

Figure 5-2 The MessageDialogDemo Project in the Designer (Source: MIT App Inventor 2)



The `Notifier` Component

The Message Dialog

In Figure 5-3 the the method takes three arguments:

- `message` — The text of the message to display. In this example, *Something happened…*
- `title` — The title to display. In this example, *Important Message*.
- `buttonText` — The text to display on the dialog box's button. In this example, *OK*.

The `Notifier` Component

Figure 5-3 The `ButtonMessage` Component's `Click` Event Handler (Source: MIT App Inventor 2)

The Click event handler calls the `Notifier1.ShowMessageDialog` method.

The `Notifier` Component

The Text Dialog

- A text dialog displays a message and provides a box (like a `TextBox`) for the user to type input.

The `Notifier` Component

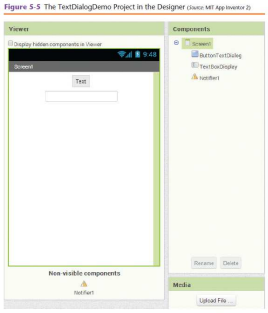
Figure 5-4 The Message Dialog Box (Source: MIT App Inventor 2)

The `Notifier` Component

The Text Dialog Contains

- A button named `ButtonTextDialog`.
- A `TextBox` named `TextBoxDisplay`.
- A `Notifier` named `Notifier1`.
- When the app runs, the user clicks the button and the text dialog shown in the image on the left in Figure 5-6 appears.

Figure 5-5 The TextDialogDemo Project in the Designer (Source: MIT App Inventor 2)



The `Notifier` Component

The Text Dialog

When the app runs, the user clicks the button and the text dialog shown in the image on the left in Figure 5-6 appears.

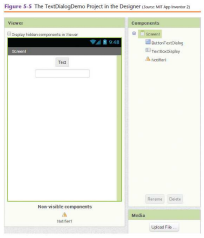


Figure 5-6 The Text Dialog Displayed (Source: MIT App Inventor 2)

The `Notifier` Component

The Text Dialog

- The Dialog prompts the user to enter his or her name.
- The user clicks the *OK* button.
- The user's input is displayed in the `TextBoxDisplay` component.

Figure 5-6 The Text Dialog Displayed (Source: MIT App Inventor 2)

The `Notifier` Component

The Text Dialog

Figure 5-7 shows the app's workspace in the Blocks Editor. Notice that the method takes three arguments:

- `message` — The text of the message to display.
- `title` — The title to display.
- `cancelable` — A true or false value.

The `Notifier` Component

The Text Dialog

As you can see in Figure 5-7, the event handler assigns the value for the response parameter to `TextBoxDisplay`'s `Text` property.

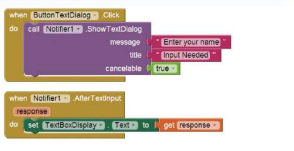
[Figure 5-7]

The `Notifier` Component

The Text Dialog

Shows the app's workspace in the Blocks Editor.

Figure 5-7 The App's Workspace in the Blocks Editor (Source: MIT App Inventor 2)



The `Notifier` Component

The Text Dialog

- The `ButtonTextDialog.Click` event handler calls the `Notifier1.ShowTextDialog` method.
- The method takes three arguments.
- `message` – The test of the message to display.
- `title` – The tile to display.
- `cancelable` – A `true` or `false` value.
  - If `true`, the dialog will have a *Cancel* button
  - If `false`, the dialog will only have an *OK* button.

The `Notifier` Component

The Text Dialog

- After the user clicks *OK* or *Cancel*, the box closes and an `AfterTextInput` event occurs.
- The event handler has a parameter named response.
- response holds the input typed by the user in to the text dialog.
- The event handler assigns the value of the response parameter to `TextBoxDisplay's Text` property.
- If the user click *Cancel*, the value of the response parameter in the `AfterTextInput` event handler will be the text *Cancel.*

The `Notifier` Component

Choose dialog

- A window that displays a `title` and a `message`.
- Lets the user click one of two buttons.
- Optionally displays a *Cancel* button.
- Displays a message and waits for the user to click a button.
- Displays a message dialog by calling the `Notifier` component's `ShowMessageDialog` method.

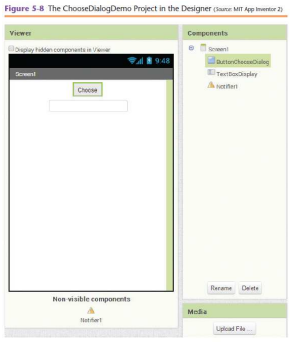
The `Notifier` Component

The Choose Dialog

•The method takes three arguments:

- `message` – The text of the message to display.
- `title` – The title to display.
- `button1Text` – The text to display on the dialog box's button.

The `Notifier` Component



The `Notifier` Component

The Choose Dialog

- The user can make a choice by clicking one of two buttons and optionally it may contain a *Cancel* button.
- The `AfterChoosing` event occurs once the user has clicked the button.
- You can create an event handler for the `AfterChoosing` event to determine which button was clicked.

The `Notifier` Component

Figure 5-9 The Choose Dialog Displayed (Source: MIT App Inventor 2)

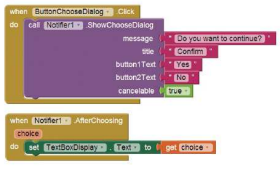
The `Notifier` Component

The Choose Dialog

- The ChooseDialogDemo Project project has
  - A button named `ButtonChooseDialog`.
  - A TextBox named `TextBoxDisplay`.
  - A Notifier named `Notifier1`.
- When the app runs, the user clicks the button and the choose dialog shown in the image on the left in Figure 5-9 appears.
- The dialog waits for the user to click the *Yes*, *No* or *Cancel* button.
- The user's choice is displayed in the `TextBoxDisplay` component.

The `Notifier` Component

Figure 5-10 The App's Workspace in the Blocks Editor (Source: MIT App Inventor 2)



The `Notifier` Component

The Choose Dialog

- The `ButtonChooseDialog.Click` event handler calls the `Notifier1.ShowChooseDialog` method and that method takes five arguments.
  - `message` – The text of the message displayed.
  - `title` – The title to display.
  - `button1Text` – The text to display on the first button.
  - `button2Text` – The text to display on the second button.
  - `cancelable` – A true or false value.

The `Notifier` Component

The Choose Dialog

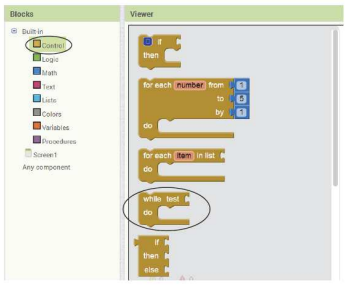
- An `AfterChoosing` event occurs once the user clicks any of the buttons on the dialog box.
- Figure 5-10 shows the event handler for the `AfterChoosing` event.
- The event handler has a parameter named `choice`.
- `Choice` holds the text of the button that the user clicked.
- The even handler assigns the value of the choice parameter to `TextBoxDisplay's` Text property.

The `while` Loop

- The `while` loop causes a statement or set of statements to repeat as long as a Boolean expression is `true`.
- The while loop has two parts
  1. A Boolean expression that is tested for a true or false value.
  2. A statement or set of statements that is repeated as long as the Boolean expression is `true`.

The `while` Loop

Figure 5-11 The `While` Loop Block (Source: MIT App Inventor 2)



The `while` Loop

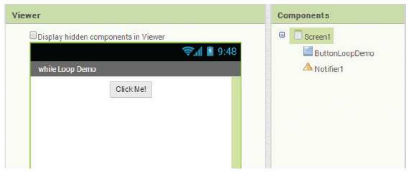
Figure 5-12 The `While` Loop (Source: MIT App Inventor 2)

The `while` loop has two sockets: `test` and `do`.

- Test Socket
  - Holds a Boolean expression.
    - If TRUE the blocks in the `do` socket are executed.
    - If FALSE the loop ends.
  - Each time the loop executes the blocks in its `do` socket, the loop is *iterating* or performing an *iteration.*

The `while` Loop

Figure 5-13 The WhileLoopDemo Project in the Designer (Source: MIT App Inventor 2)



The `while` Loop

Figure 5-13 shows the WhileLoopDemo.

- Notice the project has
  - A Button named `ButtonLoopDemo`.
  - A `Notifier` named `Notifier1`.
- When the user clicks the button in the emulator or actual device, the message dialog is shown in Figure 5-14.
- When the user clicks OK to close the dialog, another identical dialog is displayed.
- This repeats until the dialog is displayed 5 times.

The `while` Loop

Figure 5-14 Message Dialog is Displayed Five Times (Source: MIT App Inventor 2)

The `while` Loop

- Figure 5-15 shows the project's workspace in the Blocks Editor.
- This initializes the local variable named `count` to the value 0. Each time the user clicks the `ButtonLoopDemo` button, the `count` variable will keep count of the number of times the message dialog is displayed.
- The `while` loop uses the Boolean expression `count < 5`. The loop will repeat as long as the value of the count variable is less than 5.
- This block displays a message dialog.
- This block adds 1 to the `count` variable.

The `while` Loop

Figure 5-15 The Project's Workspace in the Blocks Editor (Source: MIT App Inventor 2)

The `while` Loop

The `while` Loop is a Pretest Loop

- The `while` loop tests it condition *before* performing an iteration.
- The test is done at the beginning of the loop.
- You usually have to perform some steps before the loop to ensure it executes at least one time.

The `while` Loop

The `while` Loop is a Pretest Loop

- In Figure 5-15 the first action to take place in the `ButtonLoopDemo.Click` is that the count variable is set to 0.
- If `count` was set to a value greater than 5 the loop would not have executed.
- A `while` loop will never iterate if its Boolean expression is `false` to start with.

The `while` Loop

Counter Variables

In the WhileLoopDemo app

- The `count` variable is set to the value 0.
- One is added to the `count` variable during each loop iteration.
- The loop executes as long as count is less than 5.
- The `count` variable keeps track of the number of iterations the loop has performed.

The `while` Loop

Infinite Loops

- Loops must contain a way to terminate otherwise the loop will continue to repeat until the program is interrupted.
- Infinite loops usually occur when the programmer forgets to write code inside the loop making the Boolean expression false.

The `while` Loop

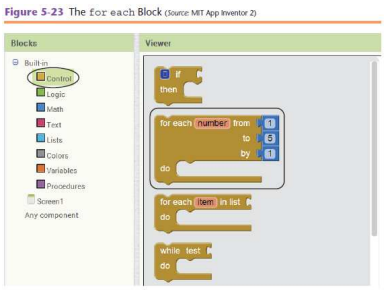
Figure 5-22 An Infinite `While` Loop (Source: MIT App Inventor 2)

The `for each` Loop

- The `for each` loop is designed to increment a counter variable over a range of values.
- It is ideally suited for problems requiring a loop that iterates a specific number of times.

The `for each` Loop

Figure 5-23 The `for each` Block (Source: MIT App Inventor 2)

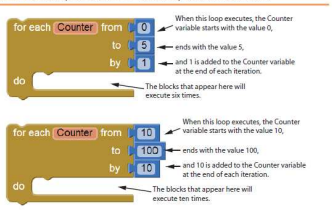


The `for each` Loop

1) The loop's counter variable – variable named number is automatically created.
2) The `from` socket – specifies the variable's starting value.
3) The `to` socket – specifies the counter variable's ending value.
4) The `by` socket specifies the amount added to the counter variable at the end of each iteration.
5) The blocks that are plugged into the `do` socket will execute each time the loop iterates.

The `for each` Loop

Figure 5-26 Examples for the `for each` Loop (Source: MIT App Inventor 2)



The `for each` Loop

- Top example the `Counter` variable starts with 0 and ends with 5.
- At the end of each iteration, 1 is added to the `Counter` variable.
- The loop will execute 6 times
- In the bottom example, the `Counter` starts with 10 and ends with 100.
- At the end of each iteration, 10 is added to the `Counter` variable.
- The loop will execute 10 times.

The `for each` Loop

Calculate a Running Total

- Programs that calculate the total of a series of numbers typically have two elements.
  1. A loop that reads each number in the series.
  2. A variable that accumulates the total of the numbers as they are read.
- The *accumulator* is the variable that is use to accumulate the total.
- It is very important that the *accumulator* starts with 0.

The `Clock` Component

- Gets the date and time from the internal system clock and provides methods and functions for working with dates and times.
- Allows you to get the current date and time from the device's internal clock.
- It also serves as a timer that performs operations at regular time intervals.

The `Clock` Component

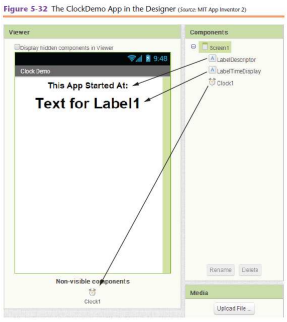
- The `Clock` component works with dates and times using a special value known as an *instant*.
- An *instant* represents a number in time.
- An *instant* contains both date and time.
- You can't print an *instant* on a screen.

The `Clock` Component

Table 5-3 Date and Time Formatting Functions (*Source:* MIT App Inventor 2)

Function	Description
call Clock1 .FormatTime instant	Formats the instant that is plugged into the `instant` socket as text describing the time only.
call Clock1 .FormatDate instant	Formats the instant that is plugged into the `instant` socket as text describing the date only.
call Clock1 .FormatDateTime instant	Formats the instant that is plugged into the `instant` socket as text describing both the date and the time.

The `Clock` Component



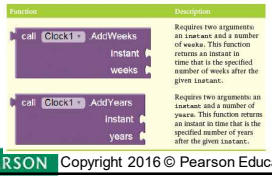
The `Clock` Component

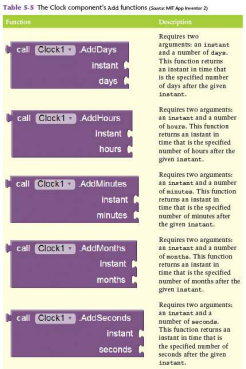
- In Figure 5-32 notice that
  - `Screen1.Initialize` event handler calls `Clock1.Now`
  - The `FormatTime` block formats the instant as text describing the time.
  - That block is assigned to the `LabelTimeDisplay.Text` property.

The `Clock` Component

Other Clock Methods

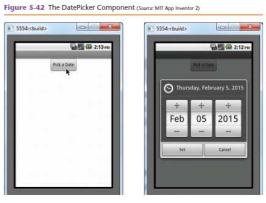
- The `Clock` component provides many other methods. Table 5-5 shows the `Clock` component's `Add` functions.





The `DatePicker` Component

The `DatePicker` component appears as a button on an app's screen. When the user clicks the `DatePicker` button it displays a dialog box that allows the user to select a date.



The `DatePicker` Component

In the Blocks Editor, there are four properties in particular that you will work with.

Table 5-6 DatePicker properties (Source: MIT App Inventor 2)

Property	Description
DatePicker1 . Day	The Day property is set to the day of the month that was last selected using the DatePicker.
DatePicker1 . Month	The Month property is set to the number of the month that was last selected using the DatePicker. The months are numbered starting with 1, so January is 1, February is 2, and so forth.
DatePicker1 . MonthInText	The MonthInText property is set to the name of the month (such as *January*, *February*, and so forth) that was last selected using the DatePicker.
DatePicker1 . Year	The Year property is set to the year that was last selected using the DatePicker.

The `DatePicker` Component

Figure 5-43 The DatePickerDemo App (Source: MIT App Inventor 2)

- When the user clicks the *Set* button in the `DatePicker`'s dialog box, an `AfterDateSet` event occurs.
- To retrieve the date, you can create an event handler for the `AfterDateSet` event.

The `DatePicker` Component

Figure 5-44 The DatePickerDemo App's Workspace in the Blocks Editor (Source: MIT App Inventor 2)

The `DatePicker` Component

- Notice that the app has one event handler: `DatePicker1.AfterDateSet`.
- It displays the name of the selected month in the `TextBoxMonthDisplay` component.
- Figure 5-45 shows an example of the app running in the emulator.

The `DatePicker` Component

Figure 5-45 The DatePickerDemo App Running in the Emulator (*Source:* MIT App Inventor 2)