6

EXPERIMENTAL EVENT

Experimental Event has a number of public methods that can be used in an experiment program. The public methods can be used in any program that imports the Experimental Event object. A number of private methods are also present, but these cannot be used by parent programs. The following sections describe the pubic methods in Experimental Event that users may employ in their programs. The headers for each section represent the method name, with the method parameters in parenthesis. This is the same general format as would appear in a program.

DeclareInput(Pin, ClockID). This method configures the Experimental Event object to be used with a digital input device and sets the event type of the object to an input event. The first parameter, Pin, is the I/O pin number connected to a digital input device. Once the pin number is provided, it is saved in Experimental Event's variable space and the direction and input registers for that pin (DIRA and INA) can be easily controlled in the background. Because the DeclareInput method sets the direction and input registers, it must be used by the cog that will control that input. The second parameter, ClockID, is used to save the location of Experimental Functions's experimental clock. The Experimental Event object uses Experimental Functions's clock to detect and debounce input events, as well as to record the duration of all event types. In order to provide the location of the experimental clock, one of Experimental Functions's StartExperiment methods must be called before the DeclareInput method. Experimental Functions's ClockID method can then be used as the ClockID parameter in the event declaration. A maximum of 151 events of any type can be declared. Figure 6.1 shows an example of declaring an input event. Once an Experimental Event object is declared as an input event, the Detect, DetectInverted, SetDebounce, State, ID, Count, SetCount, Duration, and SetDuration methods can be used to record and control the input event. Note this program also sets the system Propeller's clock speed in the constant section and includes easy to read constant representatives of event states. These instructions in the constant block will be common to most programs.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off       = 0
    Onset     = 1
    On        = 2
    Offset    = 3
    LeverPin = 5
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
```

Figure 6.1: Input declaration example program.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off       = 0
    Onset     = 1
    On        = 2
    Offset    = 3
    LeverPin = 5
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    REPEAT
        Lever.Detect
```

Figure 6.2: Detect example program.

Detect. This method detects and returns the state of an input event. The state can be read at any time using the State method (described later in this chapter). The Detect method also runs a private debounce method. The debounce feature ensures that a digital input device is active for a minimum amount of time (default 25 ms) before recording the input as on. Similarly, the device must be deactivated for the same amount of time before recording the input as off. This also means that the event's minimum duration and minimum inter-event interval are equal to the debounce duration. Typically, the Detect method will be used in a repeat loop to continually detect the input state on each program cycle. Figure 6.2 shows a typical use of the Detect method. Note, however, that nothing is done with the state returned from the Detect method. In many experiments, the Detect method will be nested within an Experimental Functions's Record method to automatically save any changes in event state during each detection (see Chapter 7).

DetectInverted. The DetectInverted method is identical to the Detect method except that it works with inverted inputs. Use this when an input circuit has an inverted signal where current flows when the input is off, and no current flows when the input is on. For example, Sharp Corporation's (Tokyo, Japan) digital infrared proximity sensors deliver current to an I/O pin when an object is not detected and stop delivering current when an object is detected. Figure 6.3 shows an example of using the DetectInverted method.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off      = 0
    Onset    = 1
    On       = 2
    Offset   = 3
    LeverPin = 5
    IRPin    = 12
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    IR:      "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    IR.DeclareInput(IRPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IR.DetectInverted
```

Figure 6.3: DetectInverted example program.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off      = 0
    Onset    = 1
    On       = 2
    Offset   = 3
    LeverPin = 5
    IRPin    = 12
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    IR:      "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    IR.DeclareInput(IRPin, EXP.ClockID)
    IR.SetDebounce(100)
    REPEAT
        Lever.Detect
        IR.DetectInverted
```

Figure 6.4: SetDebounce example program.

SetDebounce(Time). This method allows the user to modify the debounce duration used by the Detect and DetectInverted methods. The default debounce duration is 25 ms, and the maximum duration is 60,000 ms. In a sense, the debounce duration is the smallest temporal unit of data that can be detected. Therefore, the greater the debounce duration, the less accurate the data. The event duration and the inter-event interval cannot be smaller than the debounce duration. For most applications, the default debounce duration will suffice. Smaller durations may be used to increase precision, and larger durations may occasionally be required for electrically noisy input circuits. Figure 6.4 shows an example of adjusting the debounce duration for one input event.

DeclareOutput(Pin, ClockID). This method configures the Experimental Event object to be used with a digital output device and sets the event type of the object to an output event. Use of DeclareOutput is very similar to use of DeclareInput. The first parameter, Pin, is the I/O pin number connected to a digital output device. Because the DeclareOutput method sets the direction and input registers, it must be used by the cog that will control that output. The second parameter, ClockID, is used to save the location of Experimental Functions's experimental clock. As with DeclareInput, one of Experimental Functions's StartExperiment methods must be called before the DeclareOutput

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off       = 0
    Onset     = 1
    On        = 2
    Offset    = 3
    LeverPin = 5
    LightPin = 7
OBJ
    EXP:      "Experimental_Functions"
    Lever:    "Experimental_Event"
    Light:    "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
```

Figure 6.5: DeclareOutput example program.

method, and then Experimental Functions's ClockID method can then be used as the ClockID parameter in the event declaration. Once an Experimental Event object is declared as an output event, the TurnOn, TurnOff, State, ID, Count, SetCount, Duration, and SetDuration methods can be used to record and control the output event. A maximum of 151 events of any type can be declared. Figure 6.5 shows an example of using the DeclareOutput method.

TurnOn. This method turns on an output, but it does nothing if the output is already on. It also returns the event state. TurnOn can be used within an Experimental Functions's Record method to save data (see Chapter 7). Figure 6.6 shows an example where the TurnOn method is used to turn on a light output event, but only if a lever input event's state becomes an onset. Note that the event states are defined in the constant section.

TurnOff. This method turns off an output, but it does nothing if the output is already off. It also returns the event state. TurnOff can be used within an Experimental Functions's Record method to save data. Figure 6.7 shows an example where the TurnOff method is used to turn off a shock output event, but only if a lever input event's state becomes an onset.

DeclareManualEvent(ClockID). This method configures the Experimental Event object to be used as a manual event. As manual events are not necessarily related to a specific input or output device, no I/O pin parameter is needed. However, the ClockID parameter is still used to save the location of Experimental Functions's experimental clock. As with DeclareInput and DeclareOutput, one of Experimental Functions's StartExperiment methods must be

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off       = 0
    Onset     = 1
    On        = 2
    Offset    = 3
    LeverPin  = 5
    LightPin  = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Light:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        IF Lever.Detect == Onset
            Light.TurnOn
```

Figure 6.6: TurnOn example program.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off      = 0
    Onset    = 1
    On       = 2
    Offset   = 3
    LeverPin = 5
    ShockPin = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Shock:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Shock.DeclareOutput(ShockPin, EXP.ClockID)
    REPEAT
        IF Lever.Detect == Onset
            Shock.TurnOff
```

Figure 6.7: TurnOff example program.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off      = 0
    Onset    = 1
    On       = 2
    Offset   = 3
    LeverPin = 5
    ShockPin = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Shock:   "Experimental_Event"
    Latency: "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Shock.DeclareOutput(ShockPin, EXP.ClockID
    Latency.DeclareManualEvent(EXP.ClockID)
    REPEAT 20
        Shock.TurnOn
        Latency.StartManualEvent
        REPEAT UNTIL LEVER.DETECT == Onset
        Shock.TurnOff
        Latency.StopManualEvent
        WAITCNT(CLKFREQ * 30 + CNT)
```

Figure 6.8: ManualEvent example program.

called before the DeclareManualEvent method. The Experimental Functions's ClockID method can then be used as the ClockID parameter. Once an Experimental Event object is declared as a manual event, the StartManualEvent, StopManualEvent, State, ID, Count, SetCount, Duration, and SetDuration methods can be used to record and control the manual event. A maximum of 151 events of any type can be declared. Figure 6.8 shows an example of using manual events in a program. In this program, the outer repeat loop turns on shock, then starts the latency manual event. Next, the program starts another repeat loop. This loop repeatedly checks the state of the lever input event and does not stop until an onset is detected; it effectively suspends the program until a lever onset is detected. Once a lever onset is detected, the inner repeat loop ends. Next, shock is turned off, and the latency manual event is stopped. The duration of the latency event thus corresponds to latency to press the lever after shock was delivered. Finally, the program pauses for 30 seconds before starting at the beginning of

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off      = 0
    Onset    = 1
    On       = 2
    Offset   = 3
    LeverPin = 5
    LightPin = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Light:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IF Lever.State == Onset OR Lever.State == On
            Light.TurnOn
        ELSE
            Light.TurnOff
```

Figure 6.9: State example program.

the outer repeat loop. The entire process repeats 20 times. The overall result is a program that delivers 20 shocks and saves the latency to press a lever after each shock. Each lever-press also causes a 30-second break between shocks.

StartManualEvent. This method starts recording a manual event, but it does nothing if the manual event is already being recorded. It also returns the event state. StartManualEvent can be used within an Experimental Functions's Record method to save data. Use of the StartManualEvent method is very similar to that of TurnOn, however, manual events are not associated with an I/O pin, and no I/O pins are turned on as with the TurnOn method.

StopManualEvent. This method stops recording a manual event, but it does nothing if the manual event is not being recorded. It also returns the event state. StopManualEvent can be used within an Experimental Functions's Record method to save data. Use of the StopManualEvent method is very similar to that of TurnOff, however, manual events are not associated with an I/O pin, and no I/O pins are turned off as with the TurnOff method.

DeclareRawData(ClockID). This method configures the Experimental Event object to be used as a raw data event. As raw data events are not related to an

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off         = 0
    Onset       = 1
    On          = 2
    Offset      = 3
    LeverPin    = 5
    LightPin    = 7
OBJ
    EXP:      "Experimental_Functions"
    Lever:    "Experimental_Event"
    Light:    "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IF Lever.State == Onset OR Lever.State == On
            IF Lever.Count => 10
                Light.TurnOn
        ELSE
            Light.TurnOff
```

Figure 6.10: Count example program.

input or output device, no I/O pin parameter is needed. However, the ClockID parameter is still used to save the location of Experimental Functions's experimental clock. As with all declaration methods, one of Experimental Functions's StartExperiment methods must be used first in order for Experimental Functions ClockID method to be used as the ClockID parameter. The purpose of the raw data event is to ensure that Experimental Functions can save any integer data provided by the user, thus raw data events are not designed to be used with other methods in Experimental Events. A maximum of 151 events of any type can be declared.

State. This method returns the event state. For inputs, it can be used after a Detect or DetectInverted method has been called. For outputs and manual events, the State method can be used at any time. In the example in Figure 6.9, the detect method is first used to *update* the state of a lever. Then, the State method is used to *evaluate* the state of the lever. If the lever's state is onset or on, a light is turned on. Otherwise, if the lever's state is offset or off, the light is turned off. Although the Detect method does return the event state, using the

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off       = 0
    Onset     = 1
    On        = 2
    Offset    = 3
    LeverPin  = 5
    LightPin  = 7
OBJ
    EXP:    "Experimental_Functions"
    Lever:  "Experimental_Event"
    Light:  "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IF Lever.State == Onset OR Lever.State == On
            IF Lever.Count == 10
                Light.TurnOn
                Lever.SetCount(0)
        ELSE
            Light.TurnOff
```

Figure 6.11: SetCount example program.

State method here is more efficient than running the Detect method twice to compare the lever's state to both the onset and on constants.

ID. This method returns an event's ID code. Each event has a unique ID code assigned by the declare methods. The ID code is used by Experimental Functions to save data. It is not used at any other time. The ID method will be discussed further in Chapter 7.

Count. The count method returns the event count, or number of times an event has occurred in an experiment. The event count can be used in contingencies. Figure 6.10 shows an example of using the Count method. In the main repeat loop, a light is turned on only if the lever's state is onset or on, and if the lever's count is at least 10.

SetCount(NewCount). The SetCount method changes the event count to the parameter, NewCount. This can be used for a variety of purposes. For example, setting the count of a lever-press event to 0 after each reinforcer in a fixed-ratio schedule of reinforcement program allows the Count method to easily keep track

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off        = 0
    Onset      = 1
    On         = 2
    Offset     = 3
    LeverPin   = 5
    LightPin   = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Light:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IF Lever.State == Onset OR Lever.State == On
            IF Lever.Duration => 2000
                Light.TurnOn
        ELSE
            Light.TurnOff
```

Figure 6.12: Duration example program.

of the number of lever-presses required to produce a reinforcer. Figure 6.11 shows an example of the SetCount method. This program is identical to that in Figure 6.10, except that the count is set to 0 each time the light is turned on. The light therefore can only be turned on every ten lever-presses.

Duration. The Duration method returns the total time, in milliseconds, that an event has occurred during an experiment. The duration includes any ongoing events, such as a button that is currently being pressed. The total duration can be used in contingencies. Individual event duration can be created and used in contingencies by setting the duration to 0 after each instance (see the SetDuration method below). Figure 6.12 shows an example of the Duration method. Here, the light can only be turned on once the lever has been pressed for a total of 2,000 ms. This duration can be accumulated across multiple lever-presses.

SetDuration(NewDuration). The SetDuration method changes the event duration to the parameter, NewDuration. This can be used for a variety of purposes. For example, setting the duration of a lever-press event to 0 when the input state becomes an offset will allow the Duration method to return individual lever-press duration. Figure 6.13 shows an example of the SetDuration method. This program is identical to that in Figure 6.12, except that the duration is set to 0 any time the lever is in the offset or off state. The light therefore can only be turned after the lever is held for 2,000 consecutive milliseconds.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    Off         = 0
    Onset       = 1
    On          = 2
    Offset      = 3
    LeverPin    = 5
    LightPin    = 7
OBJ
    EXP:     "Experimental_Functions"
    Lever:   "Experimental_Event"
    Light:   "Experimental_Event"
PUB Main
    EXP.StartExperiment_NoData
    Lever.DeclareInput(LeverPin, EXP.ClockID)
    Light.DeclareOutput(LightPin, EXP.ClockID)
    REPEAT
        Lever.Detect
        IF Lever.State == Onset OR Lever.State => ON
            IF Lever.Duration => 2000
                Light.TurnOn
        ELSE
            Lever.SetDuration(0)
            Light.TurnOff
```

Figure 6.13: SetDuration example program.