5

PROPELLER EXPERIMENT CONTROLLER OVERVIEW

The Propeller Experiment Controller (PEC) software primarily consists of two interdependent Spin objects: Experimental Event and Experimental Functions. Both objects contain a variety of methods that allow the user to execute complex tasks using simple, concise instructions. These objects were designed to work together to fulfill the complex requirements of behavioral research, and often use a specific workflow to implement an experiment. In this chapter, we provide an overview of the PEC system, describing the overall roles of Experimental Event and Experimental Functions, and discuss the basic workflow of a typical experiment. More detailed descriptions of Experimental Event and Experimental Functions are provided in Chapter 6 and Chapter 7, respectively.

Experimental Event Overview

The **Experimental Event** object is used to detect and control events in an experiment. Each event represents a dependent or independent variable of interest and will use its own instance of the Experimental Event object. For example, consider an experiment that delivers food to a rat, contingent on the rat pressing a lever while a light is activated. This experiment program might use three Experimental Event objects, one for the food delivery, a second for the lever-press behavior, and a third for the light activation. By using three Experimental Event objects, the PEC can easily control and record data on each event.

As behavioral experiments may employ many different types of dependent and independent variables, four types of events are available in the Experimental

Event object: input events, output events, manual events, and raw data events. **Input events** are used to detect information from digital input devices, such as the pressing of a lever. Although the state of a digital input (off or on) can be represented in binary, Experimental Event uses a **four-state system** to indicate not only the off/on state of an input, but *when* the input turned off or on. At any time during an experiment, an input event exists in one of four states: *off*, the device is not currently active; *onset*, the device was recently off but has just been activated; *on*, the device was previously activated and is still being activated; and *offset*, the device was recently active but has just been deactivated.

As an example, consider an apparatus where a rat may press a lever for food. If the lever is not being pressed it is considered off. As a rat presses the lever, it is considered an onset only instantaneously, then the lever immediately transitions to the on state. The lever remains in the on state as long as the rat holds the lever. At the very moment the rat releases the lever, it enters the offset state, then immediately transitions to the off state and remains off until pressed again. The distinction between these states is important. The onset and offset states are instantaneous states that mark the start and stop time of an event. Consequences may be implemented for any of these states. For example, a food pellet may be provided for a rat only the very instant it presses a lever (the onset state). If pellets were provided during the on state, pellets would be continually dispensed as long as the rat held down the lever, potentially filling the apparatus with food. In some experiments, it may be useful to also implement contingencies during the on state, such as removing a shock when a rat initially presses a lever (the onset state) and as the rat continues to depress the lever (the on state).

In the Experimental Event object, the state of inputs is recorded as 0 (off), 1 (onset), 2 (on), and 3 (offset). To make the object easier to read, a set of constants is to represent the event states (i.e., the constant Off represents the number 0). This convention is also very useful for experiment programs. As such, many programs will build upon the basic constant block seen in Figure 5.1.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq  = 5_000_000
    Off        = 0
    Onset      = 1
    On         = 2
    Offset     = 3
```

Figure 5.1: Constant block with constants representing event states.

Figure 5.2 shows an alternative version of this constant block, with the event type constants assigned in list form in a single line.

```
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000
    #0, Off, #1, Onset, #2, On, #3, Offset
```

Figure 5.2: Alternative constant block with constants representing event states.

Inputs are also **debounced** when the states are determined. The debouncing process requires a digital input device active for a minimum amount of time (default 25 ms) before changing the input's state. Debouncing is used to prevent input events from falsely generating multiple onsets or offsets from a noisy input signal or circuit. Essentially, debouncing makes it easier for the Propeller to detect a true input event. Figure 5.3 shows a timing diagram of input event states and debouncing. The top line indicates the signal from the input device. When the input device line is high, current is flowing from the input device to the Propeller. When the input device line is low, current is not flowing from the input device to the Propeller. The bottom line indicates the state of the input event as determined by Experimental Event's methods. A high line indicates the input state is on, while a low line indicates the input state is off. The input state alters only if there is a change in current from the input device that is consistent for an entire debounce interval (25 ms). Note that the brief changes in current at 100 and 450 ms do not cause the input state to change. The input state line indicates two instances of the input event. Onsets occur only in the millisecond the state changes from off to on (225 and 425 ms), while offsets occur only in the millisecond that the state changes from on to off (325 and 625 ms).

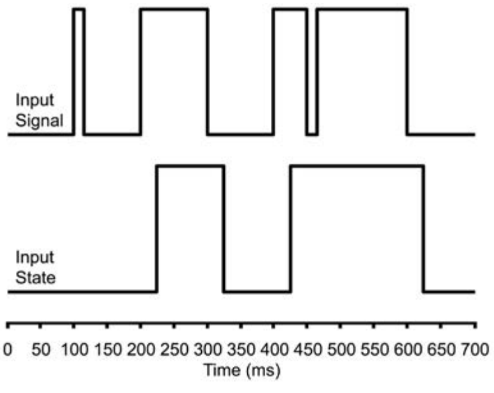


Figure 5.3: Timing diagram of input state and debouncing.

The PEC's powerful four-state system provides an efficient manner to record information about input events. In addition to the input state, the count and duration are of each input event are recorded in the background and can be read or modified at any time during an experiment. Contingencies can therefore be easily implemented on the state, count or duration of an input event.

Output events control digital output devices such as lights, feeders, fans, and motors. Output events also use the same four-state system as input events. However, no debouncing is required as the Propeller knows exactly when it changes the state of an output device. When instructed to turn an output on or off, the Propeller also updates the output event's state, count, and duration in the background, as with input events.

Manual events are used in circumstances where an event cannot be associated with a digital input or output device, but the four-state system is still desired. Often manual events are used to implement the four-state system with analog or complex devices. For example, the Propeller can generate tones by rapidly turning on and off an I/O pin connected to a speaker. The frequency the pin oscillates corresponds to the pitch of the sound. Tone generation cannot be activated in the same manner as a normal output such as a stimulus light, so the standard output event is not adequate. Instead, a manual event can be started when the tone starts playing and stopped when the tone stops. Manual events can also be used for specific temporal measures that are not associated with input or output devices, such as post-reinforcement pauses or response latency. The benefit of using manual events is that the event state, count, and duration can be read or modified in the same manner as with input and output events.

Raw data events are unique in that they have no associated state, count, or duration. As with manual events, raw data events are often used to record information about analog or complex devices, both inputs and outputs. However, raw data events are used to save a specific data value at an instantaneous point in time. For example, raw data events can be used to save integer data from analog or complex input devices such as temperature, humidity, or light levels. Raw data events can also be used to record information about output events. For example, an experiment that administers varying levels of voltage to a shock grid may benefit from recording the voltage at the moment an animal presses a lever to terminate shock. As the Propeller can communicate with an enormous variety of sensors and other devices, the PEC software does not provide methods to communicate with each device. Instead, browse Parallax's Propeller Object Exchange for objects dedicated to communicating with specific sensors. The raw data event then provides a convenient way to save recorded information about these devices in a manner similar to other event types.

General use of the Experimental Event object will require importing one instance of Experimental Functions, and up to 151 instances of Experimental Event. The 151 Experimental Event limit is a result of memory constraints of the Propeller. The limit will not likely be encountered in practice. After using a version of Experimental Functions's StartExperiment method, each instance of Experimental Event can be declared as a specific type of event: input event, output event, manual event, or raw data event. After an event is declared, several methods are available for detecting and changing the state, count, and duration of events. Chapter 6 provides a detailed description of these and other methods in the Experimental Event object.

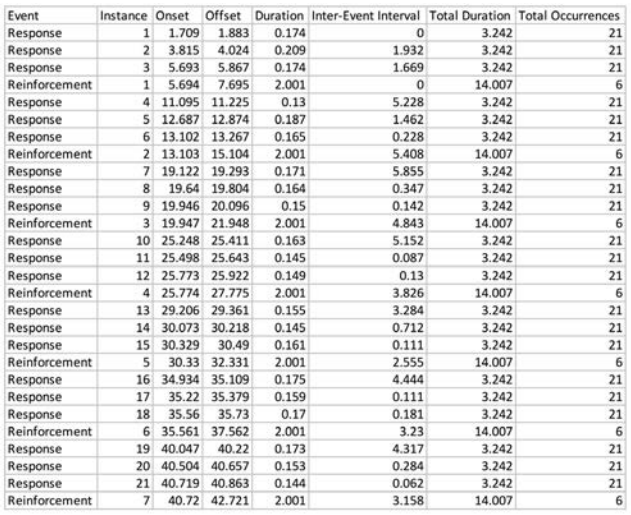
Experimental Functions Overview

The **Experimental Functions** object is used to execute common tasks that are needed by many different types of automated behavioral experiments. Although Experimental Functions can be used independently, it is designed to be used with one or more Experimental Event object. Each Experimental Event object will supply information to Experimental Functions about an event in the experiment.

Experimental Functions has three major roles in an experiment. The first major role is to maintain time using a precise **experiment clock.** The experiment clock runs in a dedicated cog, meaning the clock runs in the background and will not be affected by other code. The clock value increments each millisecond and has a maximum value of 2,147,483,647 (the maximum value of a long variable). It can run for about 596 hours, or 24 days. The experiment clock has been tested for 7 consecutive hours and is still accurate to the millisecond. Given the timing accuracy of the Propeller, longer duration tests were not conducted. The experiment clock also records the number of days that have passed since the clock was started, and the number of milliseconds that have passed in the current day. The experiment clock value can be reset at any time. It can also be changed to a specified value, such as one representing the current time of day. Setting the clock value to the current time can be useful in programs that need to execute code at specific times of the day. The Experimental Event object also uses information from the experiment clock to detect and debounce inputs, as well as to record the duration of input events, output events, and manual events.

The second major role of Experimental Functions is to record information about events as they occur during an experimental session. As the Propeller's built-in memory is limited, an SD card is used as external storage device. Experimental Functions creates a **memory file** on the SD card, with a default name of "memory.txt". The memory file is used to quickly record very basic information while the experiment is in progress. Experimental Functions collects information from each Experimental Event object in an experiment, and records an abbreviated form of this information in the memory file. The condensed format used to record information in the memory file enables this process to occur quickly during an experimental session without causing any delay in the experiment.

The third major role of Experimental Functions is to save detailed data about events after the experiment ends in a **data spreadsheet**, with a default name of "data.csv". The information recorded in the memory file is not easily comprehended; however, creating a more intelligible file would cause delays during an experiment. As a solution, Experimental Functions creates the concise memory file during an experimental session, then after the session is complete, uses the memory file to create a detailed data spreadsheet. The data spreadsheet can then be viewed in programs such as Microsoft Excel. Figure 5.4 shows an example of the data spreadsheet, exported from Excel as a pdf file. The data were generated by a fixed-ratio schedule of reinforcement program, with a session duration of 1 minute. The event column refers to the events in the program, in this case a response event, and a reinforcer event. These correspond to two Experimental Event objects in the program. The instance column refers to the number of the event, such as response instance 1, the first response. The onset and offset columns refer to the start and stop times of an event, respectively. Duration refers to the duration of a single instance of an event, while total duration refers to the combined duration of all instances of an event. Inter-event interval refers to the time between events of the same type. Note that an inter-event interval cannot be calculated for the first instance of an event. Finally, total occurrences refers to the total number of times an event occurred in a session. When the data spreadsheet is produced, it is initially sorted by event type. In most spreadsheet programs, the data can be sorted by column. In this example, the data were sorted by onset so that events appear in chronological order. Raw data events use a slightly different set of column headings and will be described in more detail in Chapter 7.



Event	Instance	Onset	Offset	Duration	Inter-Event Interval	Total Duration	Total Occurrences
Response	1	1.709	1.883	0.174	0	3.242	21
Response	2	3.815	4.024	0.209	1.932	3.242	21
Response	3	5.693	5.867	0.174	1.669	3.242	21
Reinforcement	1	5.694	7.695	2.001	0	14.007	6
Response	4	11.095	11.225	0.13	5.228	3.242	21
Response	5	12.687	12.874	0.187	1.462	3.242	21
Response	6	13.102	13.267	0.165	0.228	3.242	21
Reinforcement	2	13.103	15.104	2.001	5.408	14.007	6
Response	7	19.122	19.293	0.171	5.855	3.242	21
Response	8	19.64	19.804	0.164	0.347	3.242	21
Response	9	19.946	20.096	0.15	0.142	3.242	21
Reinforcement	3	19.947	21.948	2.001	4.843	14.007	6
Response	10	25.248	25.411	0.163	5.152	3.242	21
Response	11	25.498	25.643	0.145	0.087	3.242	21
Response	12	25.773	25.922	0.149	0.13	3.242	21
Reinforcement	4	25.774	27.775	2.001	3.826	14.007	6
Response	13	29.206	29.361	0.155	3.284	3.242	21
Response	14	30.073	30.218	0.145	0.712	3.242	21
Response	15	30.329	30.49	0.161	0.111	3.242	21
Reinforcement	5	30.33	32.331	2.001	2.555	14.007	6
Response	16	34.934	35.109	0.175	4.444	3.242	21
Response	17	35.22	35.379	0.159	0.111	3.242	21
Response	18	35.56	35.73	0.17	0.181	3.242	21
Reinforcement	6	35.561	37.562	2.001	3.23	14.007	6
Response	19	40.047	40.22	0.173	4.317	3.242	21
Response	20	40.504	40.657	0.153	0.284	3.242	21
Response	21	40.719	40.863	0.144	0.062	3.242	21
Reinforcement	7	40.72	42.721	2.001	3.158	14.007	6

Figure 5.4: An example of the standard data spreadsheet.

In addition to these three major roles, Experimental Functions also offers many other techniques that are useful to experiments but are not required by the standard workflow. For example, random number generation can be useful for randomly assigning a subject to an experimental condition or creating probabilistic outcomes such as in a variable-ratio schedule of reinforcement. Signal generation methods are another highly versatile technique included in Experimental Functions. They can be used for a variety of purposes such as creating audio tones, activating LEDs at different brightness levels, or controlling motor speed. Experimental Functions also provides methods for advanced users to read and write files to an SD card to create a customized data output format. The methods of Experimental Functions will be discussed in more detail in Chapter 7.

General Workflow of the Propeller Experiment Controller

The PEC is designed around a specific workflow that integrates Experimental Event and Experimental Functions. Figure 5.5 provides a graphic illustration of the general workflow of an experiment program. First, Experimental Functions and a single, or multiple, Experimental Events are imported. Then, Experimental Functions's StartExperiment method creates a memory file on the SD card. The experiment clock is also started at this time. Next, each Experimental Event is declared as an input event, output event, manual event, or raw data event using Experimental Event's declare methods. The program now enters the main repeat loop. This loop will vary per application of the PEC but will include everything needed for that application. It can be considered the main section of the program. Inside the loop, input events are first detected using Experimental Event's detect methods. After the state of input events is determined, other events can be affected based on the design of the experiment using a variety of methods from Experimental Event. Any changes to an event are quickly recorded in the memory file using Experimental Functions's Record method. The repeat loop is not infinite, some condition, such as session length or fixed number of trails, will cause the loop to terminate. This can be considered the end of the experiment. The activities that occur in the repeat loop, as well as the condition that causes the loop to terminate, are caused by a combination of specific instructions from Experimental Event and Experimental Functions, are covered in later chapters, as well as basic Spin logic. This section is the most free-form, and while it does require users to carefully consider what they want the program to do, it also provides substantial freedom for designing many types of experiments or other automated projects.

After the main repeat loop terminates, Experimental Functions's PrepareDataOutput method creates the first part of the data spreadsheet file on the SD card. Then, the Experimental Functions's SaveData method can be used to save detailed data about each event using the contents of the memory file. After data from every event are saved, Experimental Functions's Shutdown method unmounts the SD card, allowing the user to transfer the data spreadsheet to the computer. This basic workflow enables many types of experiments and automated projects. The user is also free to modify this workflow to fit their needs. Chapter 8 provides more detailed examples of how the PEC and this workflow are used in practice.

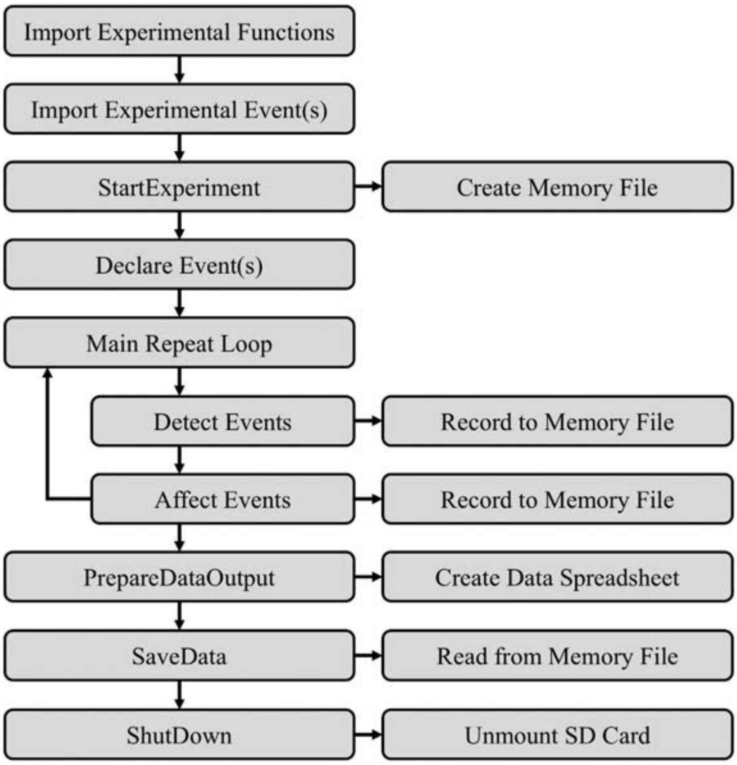


Figure 5.5: General workflow of an experiment program.