**Patch Panel Tutorial**

**What is the Patch Panel and how does it work?**

The Patch Panel is a service that runs in the background and observes the Event Heap in order to translate events. Every event that is posted to the Event Heap is a translation candidate. If a mapping is specified for an event, the Patch Panel will post the corresponding output events to the Event Heap. This functionality is especially useful for prototyping new input or output devices and interfacing these devices to different applications.

Imagine you want to create a new device that combines a sensor-augmented toy (the iDog, pictured below, can detect when it is handled) with a speaker. This new device can be prototyped by reusing existing hardware components (known as iStuff) that communicate through the Event Heap. The problem is, that the iDog and the speaker were not designed to work together; the iDog sends an iDog event but the speaker waits for a *Play* event. Of course you could change the code so that the iDog sends *Play* events or the speaker accepts *iDog* events, but with the Patch Panel you can solve this Problem without modifying source code. Instead, you can simply establish a mapping between the action (handle iDog) and the effect (*play* sound). Afterwards the Patch Panel will send a *Play* event whenever it receives an *iDog* event (it translates the *iDog* events into *Play* events).

It is also possible to map a single incoming event to a list of outgoing events, so that you can specify several different effects for the same input (similar to macro concept). For example, a button could be mapped to “start the room” by turning on the light as well as the projector.
The mechanism to programatically set mappings in the Patch Panel is through an event interface. The Patch Panel accepts PPMapping events, which contain fields to indicate the event to map, and the corresponding output events. An interesting consequence of the event interface is that mappings can contain PPMapping events as output events. Thus some mappings can change the active mappings when triggered.

You can use PPMapping Events to build a State Machine. Mappings can be viewed as states, and changes of mappings can be viewed as state transitions. The current set of mappings in the Patch Panel reflects the state of the machine. Below is an example of a finite state machine that uses a button to toggle the lights and projectors on and off. For more technical details, see the publications site of the Media Computing Group at RWTH Aachen.

http://media.informatik.rwth-aachen.de/
What is the Patch Panel Manager? – An Overview

The Patch Panel Manager is a Graphical User Interface which allows you to configure mappings in the Patch Panel. It provides three different ways to do that: the Simple Mode, the Advanced mode and a Script Language.

The **Simple mode** is quite easy to use, even if you have no idea of how the Event Heap works. You can use it to map a predefined action (e.g. “handle iDog”) to a predefined effect (e.g. “play sound”). You do not even need to know anything about programming to do so. The disadvantage of this simplicity is, that the offered functionalities are fairly limited. It is i.e. not possible to specify a PPMapping event in the Simple Mode.

The **Advanced Mode** allows you to use all the functionalities that the Patch Panel provides. In exchange it is more difficult to handle than the Simple Mode because it exposes the raw Event Heap semantics that may be too complicated for non-technical users. You need at least some basic knowledge about the Event Heap and its API. For instance you should know what an event template is (you can find information about this stuff in the Event Heap Tutorial).

Indeed the only way to configure the Patch Panel with the Simple Mode as well as with the Advanced Mode is by setting mappings. But sometimes it might get quite complex to set all the configurations you need as mappings (especially complex state machines), even though it would still be possible in the Advanced Mode. Therefore it is also possible to configure the Patch Panel with a Script Language.

You can use the **Script Language** to write a text file in which you define sets of events, states, global variables and transitions. You can import this script file to the Patch Panel Manager using the button “Import Group...” (in the Advanced Mode). The Patch Panel Manager will compile your script into mappings and show them in the Advanced Mode, so that you can change and extend them there. In many cases this will be easier than setting all the mappings in the GUI.
How to use the Simple Mode

At the left frame, called Mappings, you can see the instructions: "select an action" and "select an effect". First click on "select an action". At the beginning it may take a short while until a combo box pops up. There you choose the action which causes a special output. E.g. "If the green button is pushed do..."

Then you specify what should happen, which effect comes out, if the action occurs. E.g. "If the green button is pushed then play a sound".

If you click again on the selected action or effect, the associated parameters (depending on the effect) appear at the right frame and you have to enter the values.

The set of actions and effects are predefined by an administrator of the system. If you would like to specify new options for the simple GUI, you can modify the configuration files (/Patch Panel Manager/sources.txt and targets.txt)
How to use the Advanced Mode

At the bottom of the left frame is an “Add” button and a combo box containing the items “Group”, “Mapping”, “On Timer Event”, “Send Event”, “Set Variable Event”, “Timer Event” and “Variable”. You can choose one of these items and add it to your configurations by clicking on the button. The item will appear in the left window. You can select an item from this window by clicking on it and then set attributes for it in the right window.

Since every item has to belong to a group (except a group itself), you will first have to add at least one group before you can add other items. A group is a folder which contains some mappings belonging together. If you have added several groups and want to add a new item to one of these, you first have to select one group by clicking on it.

When you add a “Mapping” you can define the incoming event for this mapping in the right frame. But this event will still be mapped to nothing. You have to select the mapping (click on it) and add one or more other items to it, in order to make it complete. If you add “Send Event” to it, you can define the outgoing event of the mapping in the right frame. That way you will create the same kind of mapping, which is also possible to set in the Simple Mode (just map one event to another event). But in the Advanced Mode other things are possible:
- Firstly you can set a timer (by choosing “Timer Event”) and enter its delay (in milliseconds) in the right frame. So if you add this timer to a “Mapping”, the timer will start counting down whenever the incoming event is received and it can be used to trigger other mappings. A “Timer Event” can also be used to cancel a timer that was set before (you can choose between “Set Timer” and “Cancel Timer” in a combo box at the right frame).
- Secondly you can add other mappings (instead of events) to a “Mapping”. Thus you will create a PPMapping Event. That means, that these mappings are not
active yet, but they will be set as soon as the incoming event for the “Mapping” is received.
- A special kind of mapping is the “On Timer Mapping”. It is triggered by a timer which was activated by any other mapping and run out.
- In addition to that you can also define global variables within a group by choosing “Variable” in the combo box and pressing the “Add” button. Name, Type and Value of these variables can be set in the right frame.

How to write a Patch Panel Manager script

Global Variable Declaration

You can declare variables at the top of a script file. Declarations consist of type, name and value and look like this:

```
int firstTimerName = 0;
doUBLE secondTimerName = 3;
string thirdTimerName = “Play”;
```

Possible types are int, long, double, float, boolean and string (in quotes). The value must always be specified in the declaration and it has to be a constant (it can not be set based on other variable values or input events).
Of course global variables in the script will be shown as global variables in the Advanced GUI as well.

Event Declaration

You can define specifications for events and use them later in the script. Each event needs a unique name and an actual event type. So the head of an event declaration looks like the following:

```
event MyNewEvent type NewEvent
```

Each file in an event consists of type, name and post value. The first field is already given by the head of the declaration. The name of this field is always ‘EventType’ an its type is ‘string’. The value of this field is the actual event type (so in the example above it is ‘New Event’).
Further fields can be defined in the body of the event declaration.

```
event MyNewEvent type NewEvent { 

    int Counter = 6;
    string DeviceName = “Mouse”;
}
```

This example is the declaration of an event with three fields. The fields are called EventType, Counter and DeviceName. Their values are NewEvent, 6 and Mouse.
The possible types for an event field are \texttt{int}, \texttt{short}, \texttt{float}, \texttt{long}, \texttt{double}, \texttt{boolean} and \texttt{string} (in quotes).

The event declaration itself will not be shown in the Patch Panel Manager GUI, but they can be used in mappings which you can see in the GUI.

\textbf{State Declaration}

In every script at least the initial state has to be declared. It reflects the current status of the Patch Panel before any events are received. The head of it consists of the introducing words ‘initial state’ and the \texttt{name} of the state.

\begin{verbatim}
initial state MyInitialState
\end{verbatim}

Further states can be declared in the same way, just leaving out the word ‘initial’ at the beginning.

\begin{verbatim}
state StateName
\end{verbatim}

In the body of the declaration you can define what should happen when a specific event occurs (of course the event has to be declared before).

\begin{verbatim}
initial state MyInitialState {
    on event1 {
        send event2;
        set timer 500;
    }
    on event3 {
        goto StateName
    }
}
\end{verbatim}

As you can see, it is possible to send \texttt{event2} when \texttt{event1} is received. In the Advanced GUI this would be represented as a normal mapping between two events. You can also set (or cancel) a timer as a reaction to an event. This is reflected by mapping a timer event to a normal event in the GUI. Furthermore you can go to another state when you receive an event. The existence of different states and the possibility to switch between them will be translated into PPMapping events in the Advanced GUI. The mappings of the current state will always be shown as active mappings while the mappings of the other states are represented in PPMapping events.

Instead of reacting on an specified event, one could also react on a timer that runs out or on the fact that a new state is entered.
state StateName {
    on enter {
        set timer 4500;
    }

    on timer {
        goto MyInitialState;
    }
}

In the example above, a timer is set whenever the state StateName is entered. When this timer runs out (after 4500 milliseconds) the state is switched to MyInitialState.

Here you can see a complete script file:

```plaintext
int doubleClicksDone = 0;

event Input type ClickInput {
    int ButtonState = 1;
}

event DoubleClick type DoubleClickEvent {
    int DoubleClicksDone = doubleClicksDone;
    string Message = "Hello world!";
}

initial state Initial {
    on ClickInput {
        goto OneClickDone;
    }
}

state OneClickDone {
    on enter {
        set timer 500;
    }
    on ClickInput {
        doubleClicksDone = doubleClicksDone + 1;
        send DoubleClick;
        goto Initial;
    }
    on timer {
        // Timed out
        goto Initial;
    }
}
```

There are more example script files in the iStuff Framework at iStuff Framework-<version>\Patch Panel Manager\scripts.