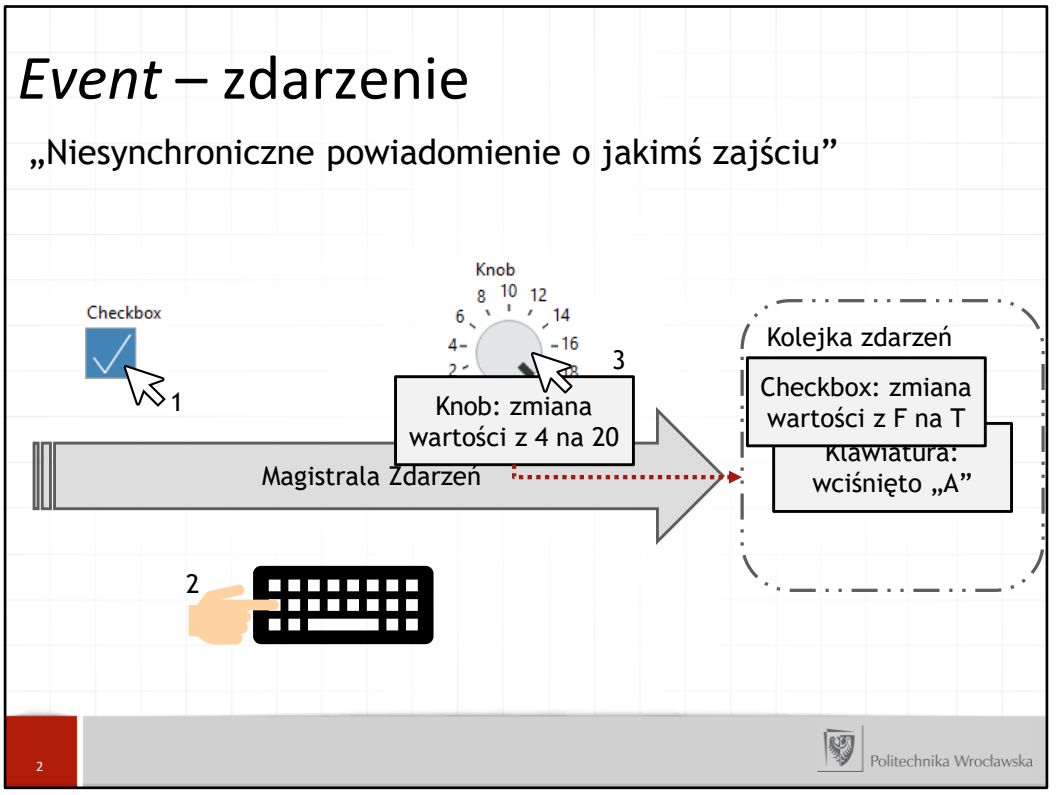
Wykład 7
19.04.2024 r.
Struktura *Event*

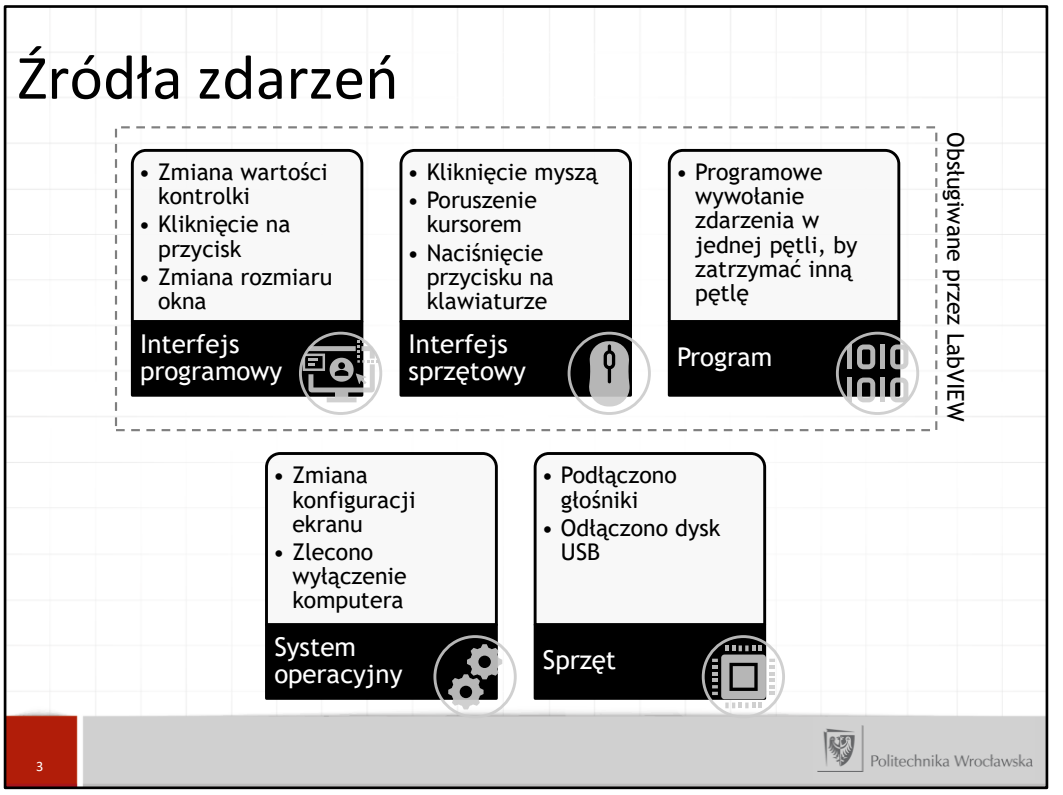
Adrian Kaim

Politechnika Wrocławska



Zdarzenie (ang. Event), to niesynchroniczne powiadomienie/zarejestrowanie zajścia jakiegoś wydarzenia lub akcji. Informacja o zdarzeniu przekazywana jest do tzw. Kolejki zdarzeń przez Magistralę Zdarzeń. Kolejka zdarzeń jest tworzona zgodnie z zasadą *First In-First Out* (*FIFO*), gdzie najstarsze zarejestrowane zdarzenie zostanie pobrane z niej jako pierwsze.

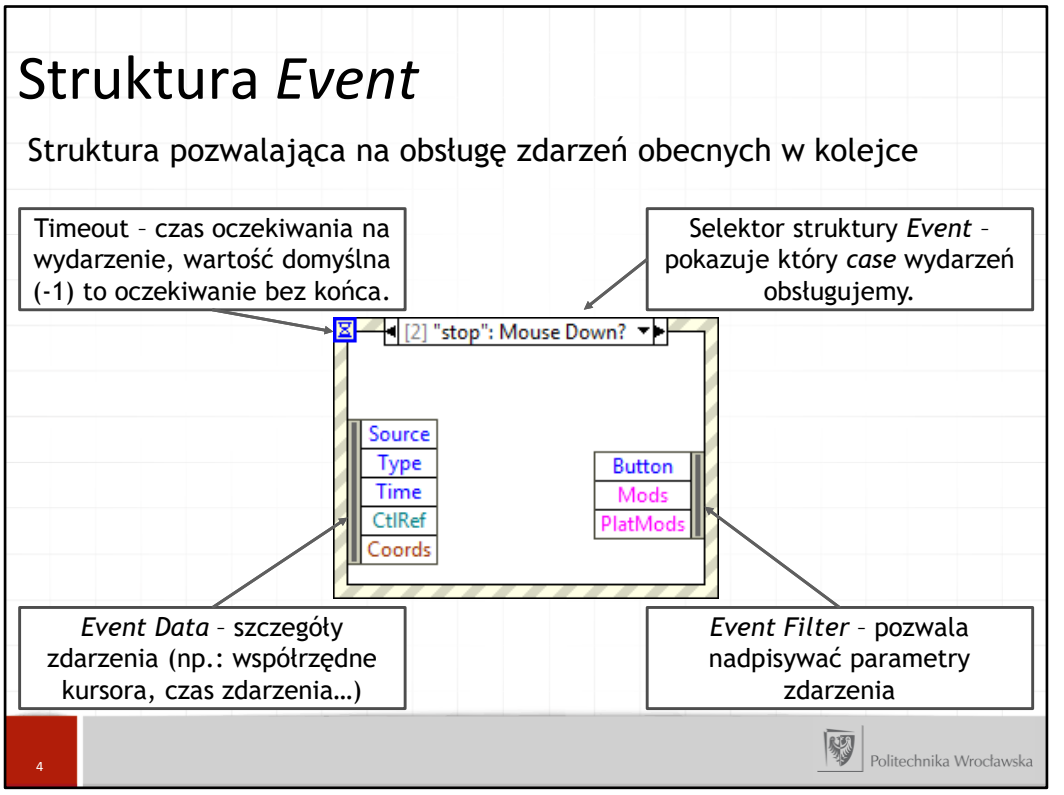
Zaletą programowania zdarzeniowego, jest niższe zapotrzebowanie na zasoby względem alternatyw takich jak cykliczne odpytywanie. Wynika to z dobrego zoptymalizowania mechanizmu magistrali zdarzeń we współczesnych systemach operacyjnych i środowiskach programistycznych.



Istnieje wiele rodzajów i źródeł zdarzeń. Do tych obsługiwanych przez LabVIEW można zaliczyć:

- Zdarzenia związane z interakcją z interfejsem programu – np.: zmiana rozmiaru okna, czy zmiana wartości kontrolki;;
- Zdarzenia związane z interfejsem sprzętowym – np.: obsługa myszy lub klawiatury;
- Zdarzenia generowane programowo – zdarzenia, które wygenerowane są dynamicznie w samym programie, np. w celu zatrzymania równoległej pętli.

LabVIEW nie obsługuje innych zdarzeń sprzętowych ani systemowych.



Struktura *Event* pozwala na obsługę zdarzeń znajdujących się w kolejce w programie. W graficznej reprezentacji przypomina strukturę *Case*. Struktura posiada:

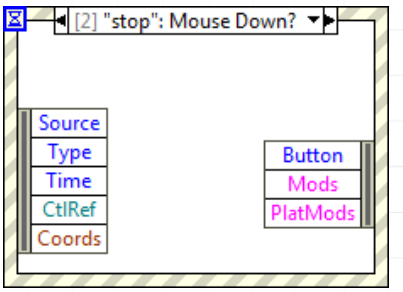
- Selektor wskazujący, który z *case*'ów zdarzeń jest w tym momencie wyświetlany;
- Terminal *Timeout*u, którym definiowany jest maksymalny czas oczekiwania na zdarzenie – jeśli po tym czasie nie wystąpi żadne zdarzenie, struktura wywoła *case* automatycznie zdefiniowany jako „*Timeout*". Domyślą wartością tego terminala jest -1, co odpowiada oczekiwaniu w nieskończoność;
- Terminale *Event Data*, udostępniające szczegóły dotyczące zdarzenia które jest obsługiwane (np.: czas zdarzenia, nowa wartość kontrolki [dla zdarzenia zmiany wartości], współrzędne kursora [dla zdarzenia związanego z myszą] itp.);
- Terminale *Event Filter*, pozwalające nadpisywać parametry zdarzenia, tak by obsłużono je niestandardowo.

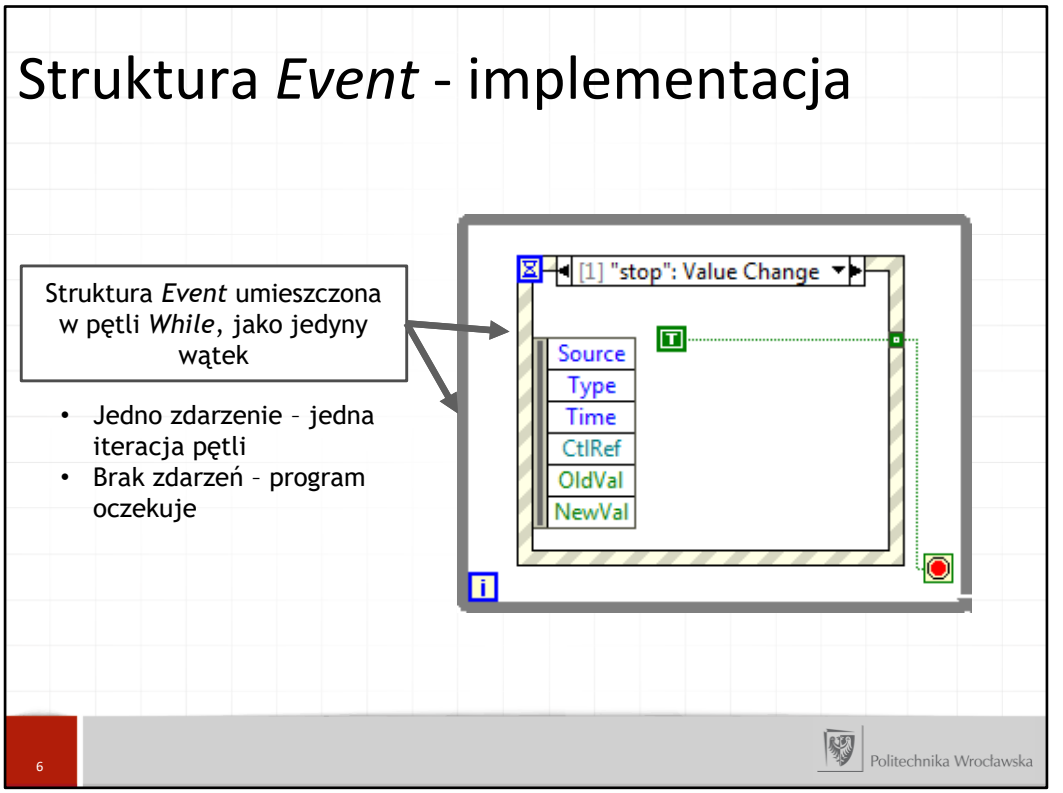
Struktura *Event* - uwagi

Kod zawarty w *Case* zdarzenia wykona się dopiero (i tylko) po wystąpieniu tego zdarzenia

Struktura obsługuje tylko po jednym zdarzeniu na raz

Każdy *Case* obsługuje inne zdarzenie (lub kilka tego samego typu)

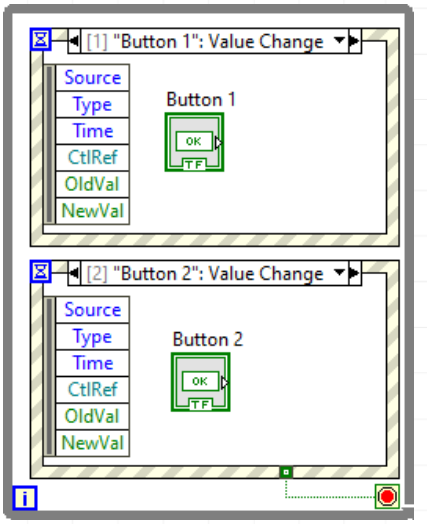




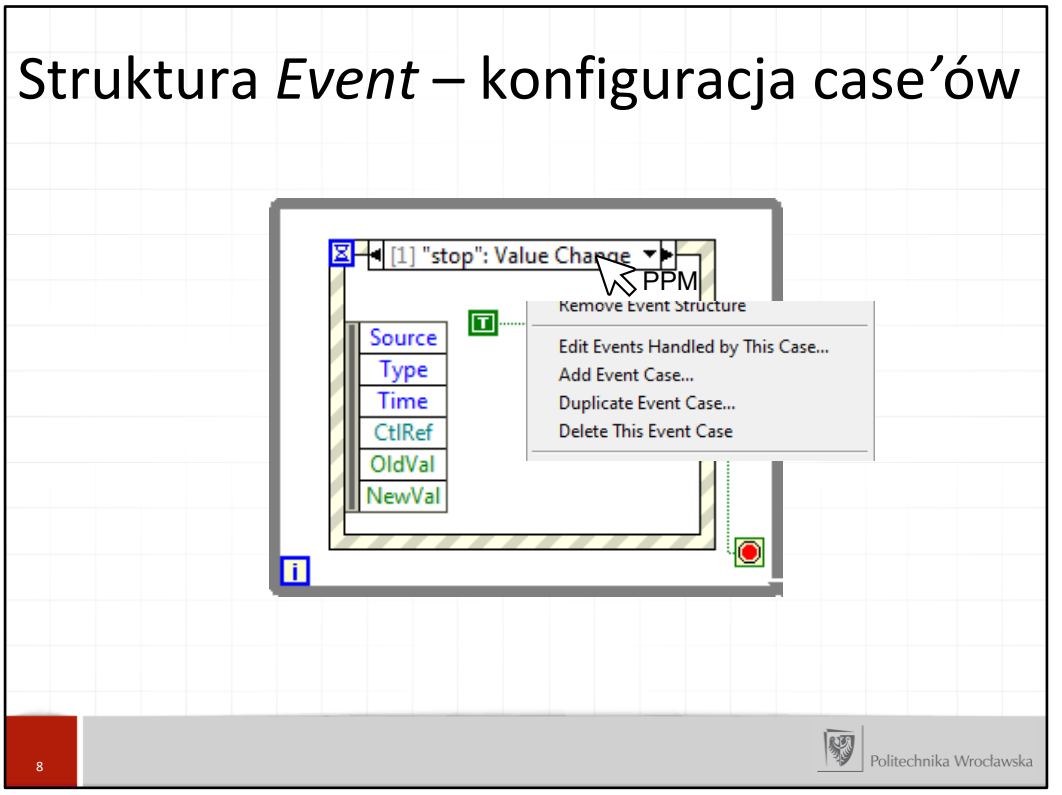
Zwykle strukturę *Event* umieszcza się w pętli *While*, co pozwala obsługiwać kolejne zdarzenia w kolejnych iteracjach, bez żadnych ograniczeń. Należy pamiętać, że zdarzenia obsługiwane są pojedynczo, dlatego nie warto umieszczać w pętli innych fragmentów kodu równolegle ze strukturą *Event* – może to niepotrzebnie spowolnić działanie programu i reakcję na zdarzenia. W przypadku gdy nie występuje żadne zdarzenie, struktura *Event* zatrzyma się na etapie wyboru odpowiedniego *case*'a i będzie oczekiwać na wystąpienie zdarzenia, które może zostać obsłużone. Jeśli czas oczekiwania przekroczy skonfigurowany *Timeout*, wybrany zostanie jego *case*.

Struktura *Event* - implementacja

Pytanie: Jak zachowa się program, gdy w jednej pętli umieszczone zostaną dwie struktury *Event*?



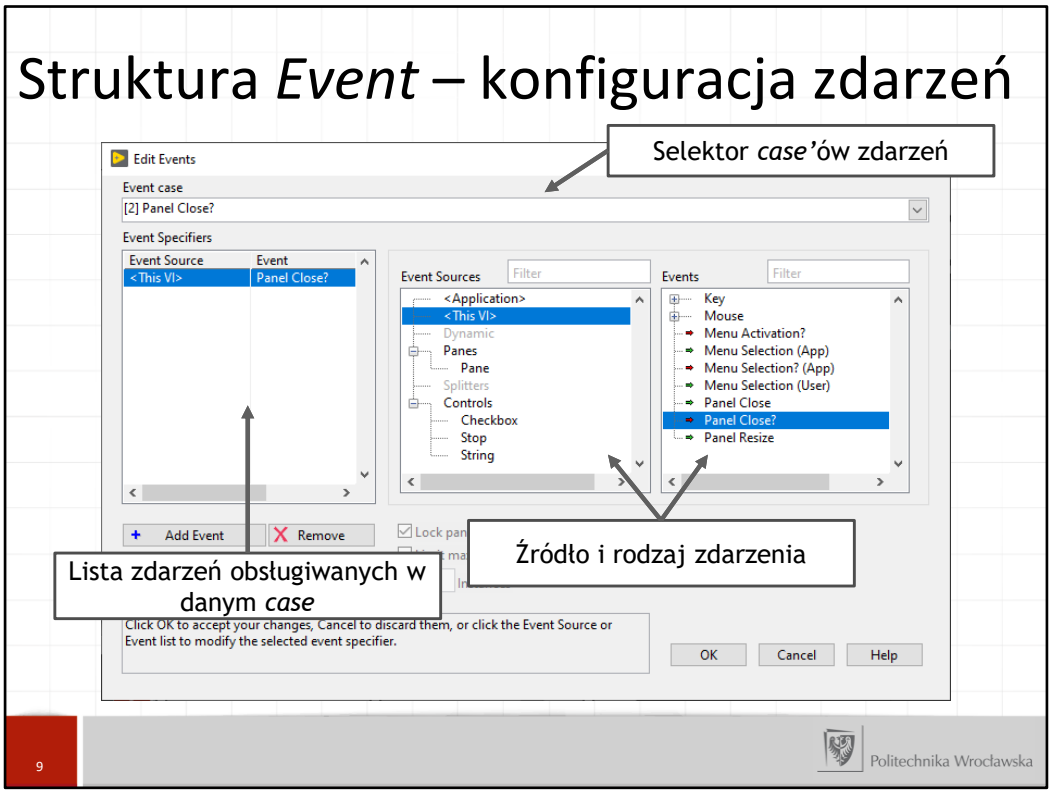
Struktura *Event* – konfiguracja case'ów



Aby skonfigurować *case*'y i obsługiwane przez nie zdarzenia, należy kliknąć prawym przyciskiem myszy na selektor struktury *Event*, a następnie wybrać:

- *Edit Events Handled by This Case* – aby zmienić, które zdarzenia spowodują wywołanie kodu zawartego w tym *case*;
- *Add Event Case…* – aby dodać nowy, pusty *case* dla nowego zdarzenia;
- *Duplicate Event Case…* – aby stworzyć *case* dla nowego zdarzenia, kopiując kod zawarty w obecnym *case*.

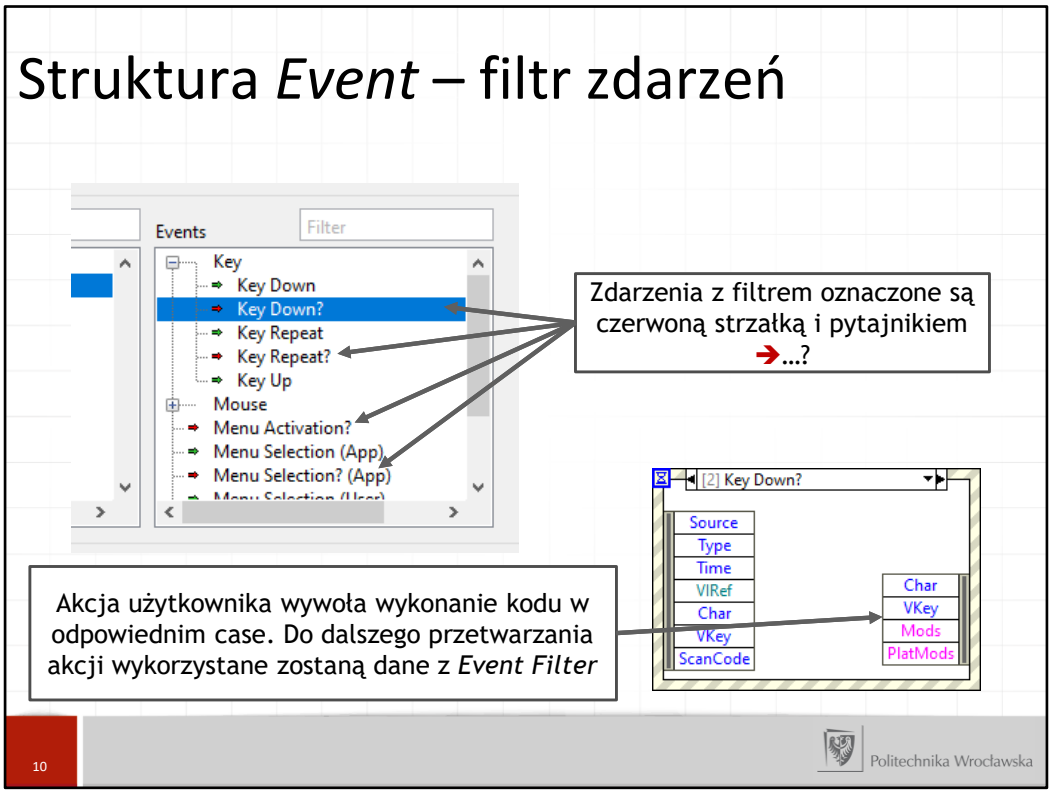
Nie ma możliwości definiowania obsługiwanych zdarzeń tekstowo, bezpośrednio w selektrorze.



Klikając jedną z opcji wymienionych na poprzednim slajdzie ukaże się okno *Edit Events*.

U góry tego okna dostępny jest selektor, pozwalający przełączać się pomiędzy wyświetlanymi *case*'ami.

Po lewej stronie widoczna jest lista zdarzeń, które będą obsługiwane przez ten *case*. Można dodawać nowe i usuwać istniejące zdarzenia korzystając z przycisków poniżej. Po prawej stronie znajdują się listy źródeł i typów zdarzeń.

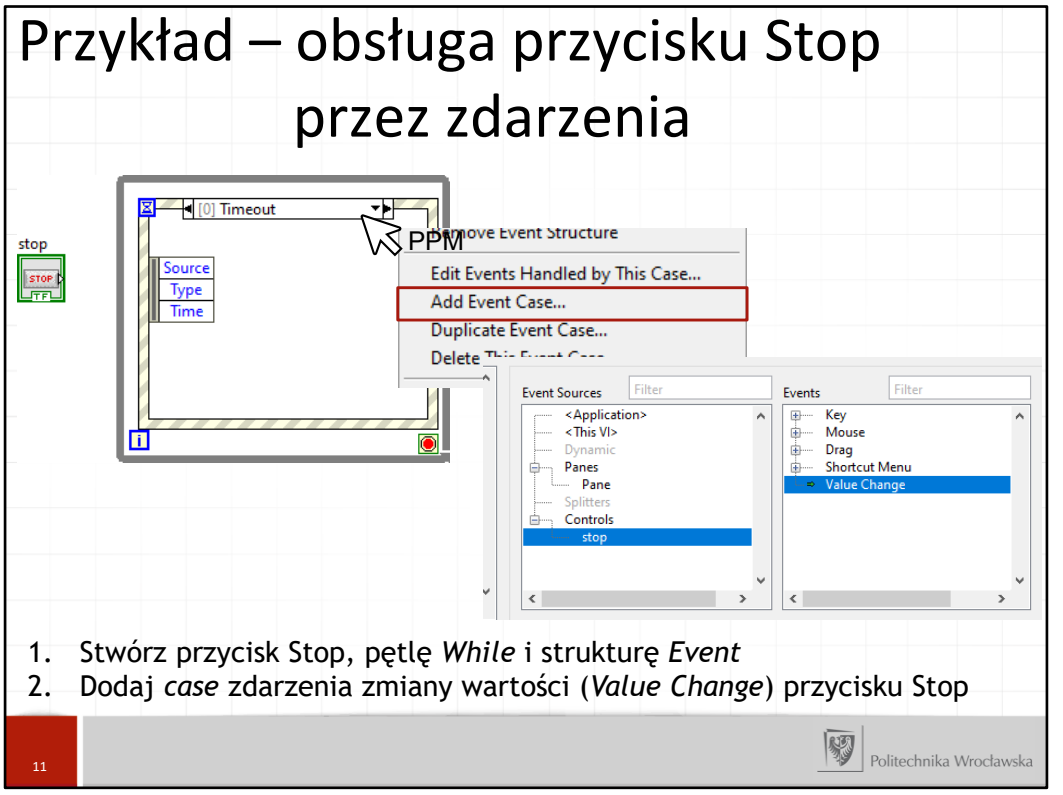


Oprócz zwykłych zdarzeń, powiadamiających jedynie o ich wystąpieniu, dostępne są też tzw. zdarzenia z filtrem. Ten rodzaj zdarzeń oznaczony jest czerwoną strzałką oraz pytajnikiem.

Gdy wykorzystane jest zdarzenie z filtrem, domyślna obsługa akcji która go wywołała jest wstrzymana do zakończenia wykonywania kodu w odpowiadającemu jej *case*. Poprzez terminale *Event Filter* mamy możliwość przekazania zmodyfikowanych danych o zdarzeniu, które zmodyfikują domyślną obsługę akcji.

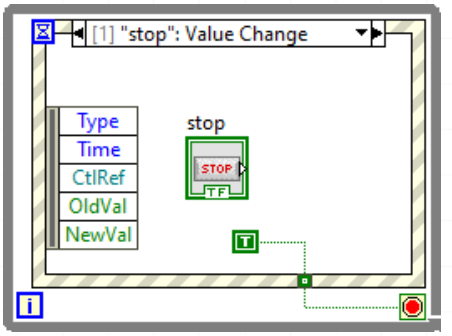
Przykłady:

1. Zdarzenie przyciśnięcia klawisza klawiatury
    - Obsługa bez filtra: znaki odpowiadające klawiszom wprowadzane są w pewnym wskazanym polu tekstowym, a struktura *Event* jest jedynie powiadamiana o tym fakcie
    - Obsługa z filtrem: wywoływany jest odpowiedni *case*, korzystając z terminali *Event Filter* możliwa jest zmiana znaków, które zostaną wpisane do wskazanego pola tekstowego.
2. Zdarzenie zamknięcia programu przyciskiem X
    - Obsługa bez filtra: okno programu zamykane jest niemal natychmiast, a struktura *Event* jest jedynie powiadamiana o tym fakcie
    - Obsługa z filtrem: wywoływany jest odpowiedni *case*, korzystając z terminali *Event Filter* możliwe jest zablokowanie zamknięcia okna programu.



1. Umieść przycisk Stop, pętlę *While* oraz strukturę *Event* na *Block Diagramie*.
2. Dodaj nowy *case* dla zdarzenia odpowiadającego kliknięciu przycisku Stop:
    a. Kliknij prawym przyciskiem myszy na selektor struktury *Event* i wybierz *Add Event Case…,*
    b. Jako źródło zdarzenia wybierz z listy kontrolkę – przycisk Stop,
    c. Wskaż zdarzenie *Value Change*.
    d. Zatwierdź przyciskiem OK

Przykład – obsługa przycisku Stop przez zdarzenia

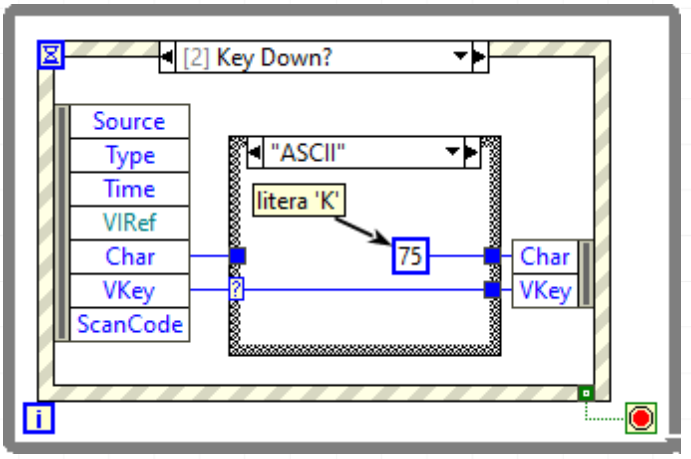


3. Dodaj kod potrzebny do zatrzymania pętli
4. Opcjonalnie: umieść terminal kontrolki Stop wewnątrz *case*'a tego zdarzenia

3. Wewnątrz *case* umieść kod potrzebny do zatrzymania pętli. Chcemy by pętla zatrzymała się niezależnie od tego, czy przycisk zmieni wartość na prawda, czy na fałsz – wystarczy zatem wykorzystać stałą o wartości prawda i połączyć ją z warunkiem zatrzymania pętli.

4. (Opcjonalnie) Umieść terminal kontrolki Stop wewnątrz *case* – w zależności od konfiguracji akcji mechanicznej tego przycisku, pozwoli to na samoczynne „odskoczenie" przycisku, gdy jego wartość zostanie odczytana (odczytana wartość nie jest nigdzie przekazywana – żaden przewód nie jest przyłączony).

Przykład – filtr klawiszy klawiatury



13

Politechnika Wrocławska

W programie widocznym na slajdzie, wszystkie klawisze odpowiadające znakom ASCII zostaną zamienione na literę ‚K'.

Zadanie 1

Stwórz program, na którego Front Panelu umieścisz dwa przyciski („Losuj” i „Stop”) oraz indykator numeryczny. Niech naciśnięcie przycisku „Losuj” spowoduje wylosowanie liczby z przedziału 0-13 i wyświetlenie jej w indykatorze, a naciśnięcie przycisku „Stop” zatrzyma program.

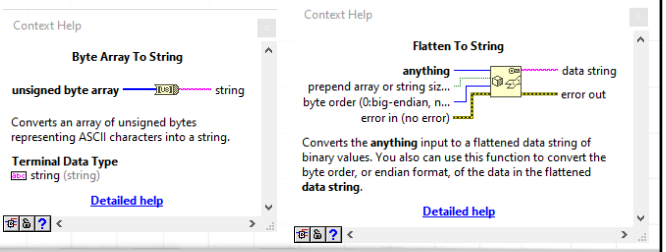
Zadanie 2

Stwórz program, który wykorzystując strukturę *Event*, będzie śledzić położenie kursora i wyświetlać jego współrzędne na wykresie.

Zadanie 3

Stwórz program, który wykorzystując strukturę *Event*:

1. będzie wyświetlać wszystkie naciśnięte klawisze w indykatorze typu string (można ograniczyć się do liter i cyfr);
2. Umożliwi usuwanie ostatniego znaku przyciskiem Backspace oraz czyszczenie historii znaków



Politechnika Wrocławska

Zadanie 4

Stwórz program, który przy próbie zamknięcia programu za pomocą przycisku x wyświetli komunikat proszący użytkownika o wciśnięcie przycisku stop.