Trigger state storage as union of structs

Ilnur Gabdrakhmanov

JINR, VBLHEP

Dubna June 29, 2020

Main objectives:

- Save trigger logic in the compact and universal form
- Trigger states must be comparable to each other
- Flexible for extension
- Working for any logical expression

Bit scheme: Period 7 BM@N

```
struct __attribute__ ((packed)) BmnTrigStructPeriod7SetupBMN
{
    bool BC1:1;
    bool BC2:1;
    bool BC3:1;
    bool VETO:1; ///<- means not VETO
    UShort_t :8; // not used
    UShort_t ThrBD:5;
    UShort_t ThrSI:5;
    UShort_t :10; // not used
};
```

BIT №	0	1	2	3	4..11	12..16	17..21	22..31
Name	BC1	BC2	BC3	VETO		BD	SI	
Length	1	1	1	1	8	5	5	10

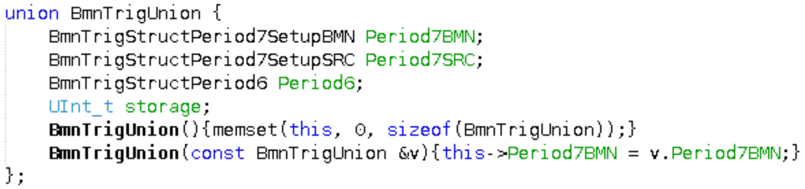
Bit scheme: Period 7 SRC

```
struct __attribute__ ((packed)) BmnTrigStructPeriod7SetupSRC
{
    bool BC1:1;
    bool BC2:1;
    bool BC3Hi:1;///<- means not BC3-Hi
    bool VETO:1; ///<- means not VETO
    bool X1:1;
    bool Y1:1;
    bool X1Y1_and_X2Y2:1; ///<- 1 - and, 0 - or
    bool X2:1;
    bool Y2:1;
    UShort_t :3; // not used
    UShort_t ThrBD:5;
    UShort_t ThrSI:5;
    UShort_t :10; // not used
};
```

Bit scheme: Period 6

```
struct __attribute__ ((packed)) BmnTrigStructPeriod6
{
    bool BC1:1;
    bool BC2:1;
    bool T0:1;
    bool VETO:1; ///<- means not VETO
    UShort_t :8; // not used
    UShort_t ThrBD:5;
    UShort_t :15; // not used
};
```

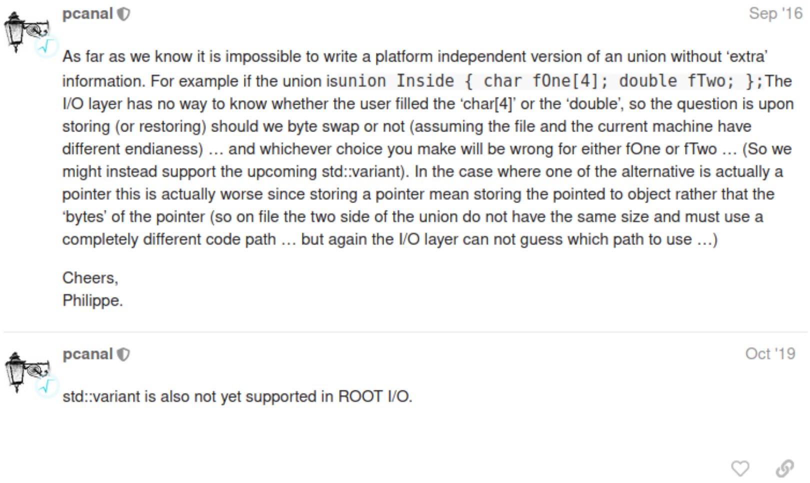
Union for structs



```
union BmnTrigUnion {
    BmnTrigStructPeriod7SetupBMN Period7BMN;
    BmnTrigStructPeriod7SetupSRC Period7SRC;
    BmnTrigStructPeriod6 Period6;
    UInt_t storage;
    BmnTrigUnion(){memset(this, 0, sizeof(BmnTrigUnion));}
    BmnTrigUnion(const BmnTrigUnion &v){this->Period7BMN = v.Period7BMN;}
};
```

- √ Volume = 32 bits as all of the fields
- √ Any future setup structs can be added and read by
- × Cannot be saved in a ROOT tree due to a streamer problem

No ROOT streamer for union and std::variant



pcanal Sep '16

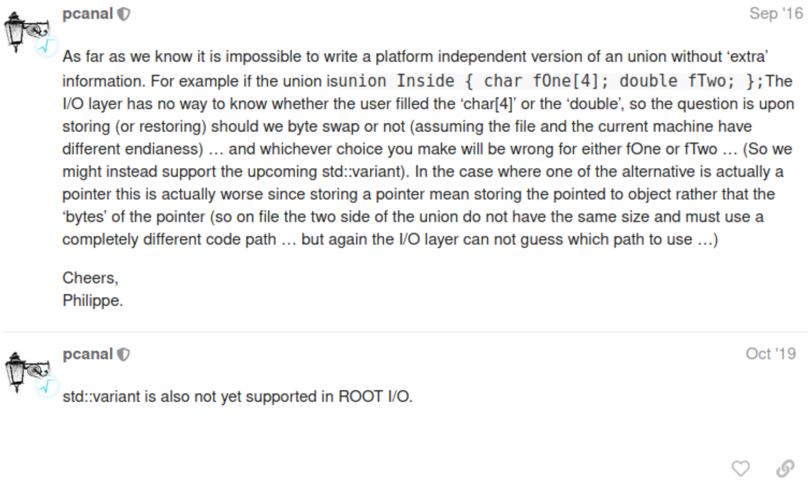
As far as we know it is impossible to write a platform independent version of an union without ‘extra’ information. For example if the union is`union Inside { char fOne[4]; double fTwo; };`The I/O layer has no way to know whether the user filled the ‘char[4]’ or the ‘double’, so the question is upon storing (or restoring) should we byte swap or not (assuming the file and the current machine have different endianess) … and whichever choice you make will be wrong for either fOne or fTwo … (So we might instead support the upcoming std::variant). In the case where one of the alternative is actually a pointer this is actually worse since storing a pointer mean storing the pointed to object rather that the ‘bytes’ of the pointer (so on file the two side of the union do not have the same size and must use a completely different code path … but again the I/O layer can not guess which path to use …)

Cheers,
Philippe.

pcanal Oct '19

std::variant is also not yet supported in ROOT I/O.

No ROOT streamer for union and std::variant



pcanal Sep '16

As far as we know it is impossible to write a platform independent version of an union without 'extra' information. For example if the union is`union Inside { char fOne[4]; double fTwo; };`The I/O layer has no way to know whether the user filled the 'char[4]' or the 'double', so the question is upon storing (or restoring) should we byte swap or not (assuming the file and the current machine have different endianess) … and whichever choice you make will be wrong for either fOne or fTwo … (So we might instead support the upcoming std::variant). In the case where one of the alternative is actually a pointer this is actually worse since storing a pointer mean storing the pointed to object rather that the 'bytes' of the pointer (so on file the two side of the union do not have the same size and must use a completely different code path … but again the I/O layer can not guess which path to use …)

Cheers,
Philippe.

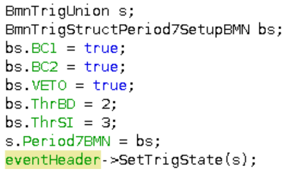
pcanal Oct '19

std::variant is also not yet supported in ROOT I/O.

Thus storage of union is implemented via UInt_t (also 4 bytes)

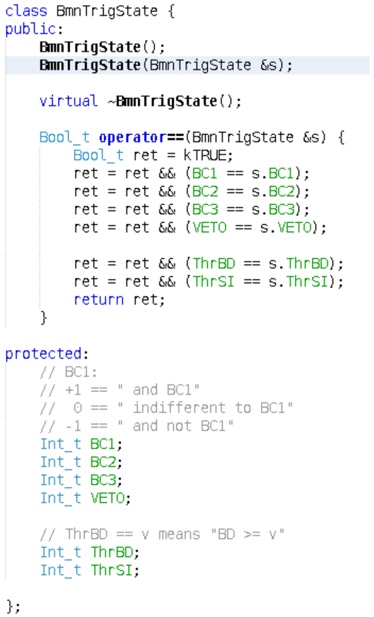
Example:

$$BC1 \wedge BC2 \wedge \overline{VETO} \wedge (BD >= 2) \wedge (SI >= 3)$$



```
BmnTrigUnion s;
BmnTrigStructPeriod7SetupBMN bs;
bs.BC1 = true;
bs.BC2 = true;
bs.VETO = true;
bs.ThrBD = 2;
bs.ThrSI = 3;
s.Period7BMN = bs;
eventHeader->SetTrigState(s);
```

Backup Slides



```
class BmnTrigState {
public:
    BmnTrigState();
    BmnTrigState(BmnTrigState &s);

    virtual ~BmnTrigState();

    Bool_t operator==(BmnTrigState &s) {
        Bool_t ret = kTRUE;
        ret = ret && (BC1 == s.BC1);
        ret = ret && (BC2 == s.BC2);
        ret = ret && (BC3 == s.BC3);
        ret = ret && (VETO == s.VETO);

        ret = ret && (ThrBD == s.ThrBD);
        ret = ret && (ThrSI == s.ThrSI);
        return ret;
    }

protected:
    // BC1:
    // +1 == " and BC1"
    //  0 == " indifferent to BC1"
    // -1 == " and not BC1"
    Int_t BC1;
    Int_t BC2;
    Int_t BC3;
    Int_t VETO;

    // ThrBD == v means "BD >= v"
    Int_t ThrBD;
    Int_t ThrSI;

};
```

Beam Counters:

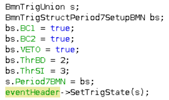
$$\begin{cases} \wedge BC2 & BC2 == 1 \\ indifferent & BC2 == 0 \\ \wedge \overline{BC2} & BC2 == -1 \end{cases}$$

BD and SI triggers:

$$\begin{cases} \wedge (BD >= 3) & ThrBD == 3 \\ indifferent & ThrBD == 0 \end{cases}$$

Example:

$$BC1 \wedge BC2 \wedge \overline{VETO} \wedge (BD >= 2) \wedge (SI >= 3)$$



```
BmnTrigUnion s;
BmnTrigStructPeriod7SetupBMN bs;
bs.BC1 = true;
bs.BC2 = true;
bs.VETO = true;
bs.ThrBD = 2;
bs.ThrSI = 3;
s.Period7BMN = bs;
eventHeader->SetTrigState(s);
```