

VIEWFILE(5)

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1	SYNOPSIS	1
2	DESCRIPTION	3
3	VIEW FILE ELEMENTS	4
4	VIEW FLAGS	5
5	EXAMPLE	6
6	BUGS	7
7	SEE ALSO	8
8	COPYING	9

Chapter 1

SYNOPSIS

Formal specification:

```
# First view in the file
VIEW VIEWNAME

FLD_TYPE      C_NAME      UBF_NAME    COUNT      FLAGS      SIZE      NULL_VAL
...
FLD_TYPE_N    C_NAME_N    UBF_NAME_N  COUNT_N    FLAGS_N    SIZE_N    NULL_VAL_N

END

...

# N view in the view file
VIEW VIEWNAME_N

...

END
```

Example view: (test.v):

```
VIEW MYVIEW1
#type      cname      fbname      count      flag      size      null

short      tshort1    T_SHORT_FLD      1      FS      -      2000
short      tshort2    T_SHORT_2_FLD    2      FSC     -      2001
short      tshort3    T_SHORT_3_FLD    3      FSC     -      -
short      tshort4    -                1      N       -      NONE

long       tlong1     T_LONG_FLD       1      FS      -      0
int        tint2     T_LONG_2_FLD     2      FS      -      0
int        tint3     -                1      -       -      -1
int        tint4     -                2      -       -      -1

char       tchar1     T_CHAR_FLD       1      F       -      '\n'
char       tchar2     T_CHAR_2_FLD     5      SC      -      'A'
char       tchar3     T_CHAR_3_FLD     2      CN      -      -

float      tfloat1    T_FLOAT_FLD      4      FS      -      1.1
float      tfloat2    T_FLOAT_2_FLD    2      S       -      -
float      tfloat3    -                1      N       -      9999.99

double     tdouble1   T_DOUBLE_FLD     2      FS      -      55555.99
```

```

double    tdouble2    T_DOUBLE_2_FLD    1    F    -    -999.123

string    tstring0    -    3    -    18    '\n\t\f\\\''\vHELLOWORLD'
string    tstring1    T_STRING_FLD    3    FSP    20    'HELLO WORLDDB'
string    tstring2    T_STRING_2_FLD    3    FCL    20    -
string    tstring3    T_STRING_3_FLD    4    FSCL    20    'TESTEST'
string    tstring4    -    1    P    15    'HELLO TEST'
string    tstring5    -    1    -    15    'MEGA TEST'

carray    tcarray1    T_CARRAY_FLD    1    FS    30    '\0\n\t\f\\\''\vHELLOWORLD ←
,
carray    tcarray2    T_CARRAY_2_FLD    1    LFSP    25    '\0\n\t\f\\\''\vHELLOWORL\ ←
n'
carray    tcarray3    T_CARRAY_3_FLD    10    FSCLP    30    '\0\\\nABC\t\f\''\vHELLO'
carray    tcarray4    -    1    -    5    'ABC'
carray    tcarray5    -    1    -    5    -

END

```

Compiled header output (C language mode):

```

struct MYVIEW1 {
    short    tshort1;        /* null=2000 */
    short    C_tshort2;
    short    tshort2[2];    /* null=2001 */
    short    C_tshort3;
    short    tshort3[3];
    short    tshort4;        /* null=NONE */
    long     tlong1; /* null=0 */
    int      tint2[2];        /* null=0 */
    int      tint3; /* null=-1 */
    int      tint4[2];        /* null=-1 */
    char     tchar1; /* null="\n" */
    short    C_tchar2;
    char     tchar2[5];        /* null="A" */
    short    C_tchar3;
    char     tchar3[2];
    float     tfloat1[4];    /* null=1.1 */
    float     tfloat2[2];
    float     tfloat3;        /* null=9999.99 */
    double    tdouble1[2];    /* null=55555.99 */
    double    tdouble2;        /* null=-999.123 */
    char     tstring0[3][18];    /* null="\n\t\f\\\''\vHELLOWORLD" */
    char     tstring1[3][20];    /* null="HELLO WORLDDB" */
    short     C_tstring2;
    unsigned short    L_tstring2[3];
    char     tstring2[3][20];
    short     C_tstring3;
    unsigned short    L_tstring3[4];
    char     tstring3[4][20];    /* null="TESTEST" */
    char     tstring4[15];    /* null="HELLO TEST" */
    char     tstring5[15];    /* null="MEGA TEST" */
    char     tcarray1[30];    /* null="\0\n\t\f\\\''\vHELLOWORLD" */
    unsigned short    L_tcarray2;
    char     tcarray2[25];    /* null="\0\n\t\f\\\''\vHELLOWORL\n" */
    short     C_tcarray3;
    unsigned short    L_tcarray3[10];
    char     tcarray3[10][30];    /* null="\0\\\nABC\t\f\''\vHELLO" */
    char     tcarray4[5];    /* null="ABC" */
    char     tcarray5[5];
};

```

Chapter 2

DESCRIPTION

View files (usually with extension `.v`) describe the data block or structure which later is generated by `viewc(8)` binary. The view compiler generate C header file for view file. Also the compiler generate the *object-file* with extension `.V`. The object-file basically is the same view file except it contains platform specific meta data. For each platform (OS/CPU changes), view file shall be re-compiled. However during the middleware operations, the message view message produced on one platform is transferable on the other platform. This basically is the core view feature, so that programmer may operate with C structures, but the middleware ensures that structure is successfully delivered to the target system in cross platform way.

View files may describe single element or array of the element (e.g. `tshort1`). The arrays are supported for `STRING` and `CARRAY` fields too. In those cases those are two dimensional arrays.

For `STRING` and `CARRAY` types, length can be specified. The length is given in total number of bytes. So if you want to store "ABC", you will need to specify *count* 4, one extra byte for EOS (zero terminator).

The next section describes each of the view file elements.

FLD_TYPE C_NAME FB_NAME COUNT FLAGS SIZE NULL_VAL

Chapter 3

VIEW FILE ELEMENTS

FLD_TYPE

Field type. This is how the field will be presented in the C structure. Following types are supported: **short**, **long**, **char**, **float**, **double**, **string** and **carray**. Carray is a blob, which might contain binary zero byte (0x00).

C_NAME

Field name in c structure. The rule against the field name applies the standard ones which applies to C variable names. **viewc** accepts max length of the 256 for the C identifier. But actual size is determined by compiler.

UBF_NAME

Optional UBF buffer field name. This is used by **Bvstof(3)** - convert C struct to UBF buffer. And by **Bvftos(3)** - convert UBF buffer to c structure. If the setup of the UBF field is not needed, then field can be skipped with "-" (minus sign in view file). In that case no mappings are processed. During the **Bvstof** and **Bvftos** invocation. If field is set, then **FLAGS** field changes the logic how the mapping is processed.

COUNT

This array element count. As in view file each of the c fields can be array, this defines the behavior. If field is set to **1**, then field is created as a normal one. If count > 1, then array is defined. The count cannot be less than 1, in such case view compiler will give an error. For **CARRAY** and **string** fields, count greater than 1 will create basically two dimensional array, where one dimension is count and another dimension is size.

FLAGS

There are defined several flags for the view file. For values see the section below. If not flags are set, then value "-" shall be used in view descriptor. Some options of the flags can be changed in the runtime with **Bvopt(3)** function.

SIZE

Number of bytes for **string** or **carray** definition. For other field types it is not valid to be set and will cause **viewc** complication error. For **string** and **carray** it is mandatory to be set size set. For other types use value "-".

NULL_VAL

This is NULL value for the field. The NULL value indicates the value when it can be considered that field is empty. The NULL value is used for following cases: 1) When transferring data from C to UBF buffer by **Bvstof(3)**. Each field is tested against NULL value. If field is NULL and the updated mode **BUPDATE**, then field is not copied to target buffer. In case of multiple occurrences, the occurrence will be lost if next C array element is not NULL. To use default NULL specification, use "-" field. Which means 0 for short/long/int. 0.0f for float/double, "" - for string and zero length carray. To encode special symbols, spaces in the NULL value, use single or double quoted string. The view compiler recognizes following escape sequences \0 - for 0x00 byte, \n, \ , \f, \ , \ , \ , \ , \ , \ , \ . For string NULLs the EOS terminator 0x00 byte is not needed. NULL values are used during the **tpcalloc(3)** operation. The buffer is initialized with default values by help of the **Bvsinit(3)** function. To set individual field to the NULL value use **Bvselinit(3)**. To test the field for NULL value use **Bvnull(3)**. If no NULL value is present for a field use keyword NONE without quotes. In that case no value will be considered NULL. This will mean that any value will be transferred to UBF. And any value will be copied over to destination service when doing **tpcall(3)**.

Chapter 4

VIEW FLAGS

F

Map the field from C struct to UBF (will be processed by **Bvstof(3)**).

S

Map the field from UBF to C struct (will be processed by **Bvftos(3)**).

N

Do not perform UBF mappings at all (**Bvstof(3)**, **Bvftos(3)** will skip the field in the same way as "-" value used for flags.).

C

Flag means generate count indicator for the field. The count field type is *short*. The count field is made as *C_* prefix for the C field name. If other field is named as combination of *C_* and this field name, then C compiler will generate error as duplicate member of the structure. The count field participates during the data transfer to UBF buffer via **Bvstof(3)**, **Bvftos(3)** commands. For the transfer for to the UBF, the count will indicate occurrences to transfer. For example 0, will make zero transfer (buffer will not be setup). The count field participates also in structure transfer view **tpcall(3)** or other buffer related IPC operation. The actual data transfered to the target system will depend on count field. The other elements in target system will be initialized with NULL value, defined in view. If count field is not set (no **C** flag present), all occurrences will be sent to the target system. And all occurrences will be copied to the UBF buffer. When data is transfered from UBF to C structure, then count field is set to number occurrences are written in the structure. For sample reference, see *C_tshort2* field and its original *tshort2* definition.

L

If this flag is set, then additional length indicator(s) are generated. This is used to indicate the actual length for the **CARRAY** data, to be written to UBF buffer. If data is converted fro UBF buffer to C structure, then field will indicate the number of bytes copied to C structure. In case of strings, the field is updated only in case when data is copied from UBF to C struct, in that case it is set to number of bytes written to C struct. When converting from C to UBF, for strings this length value is ignored and EOS termination of the string are used for length indication. The length field is produced as prefix *L_* for the given field. In the same way as for **C**, there might be conflict with duplicate field if another field is already as this field prefixed with *L_*. The field type is *unsigned short*. IF **COUNT** is greater than 1, then length indicators are array of *unsigned short*, because the length indication must be processed for each of the elements in array. If **L** flag is not set, then for **carray** full buffer size is copied to UBF and back. The length plays similar role for C struct data transfer over the **tpcall(3)** and other IPC operations. Only the number set it length are transfered to target service for the **CARRAY** fields.

Chapter 5

EXAMPLE

See `atmitest/test040_typedview` for sample usage.

Chapter 6

BUGS

Report bugs to support@mavimax.com

Chapter 7

SEE ALSO

viewc(8), Bvftos(3), Bvstof(3), Bvsinit(3), Bvselinit(3), Bvnull(3), Bvopt(3), Bvrefresh(3)

Chapter 8

COPYING

© Mavimax, Ltd