

Instructions for `ddg` (DDL Data Generator) program

By Ervin Dénes

20 March 2005

Calling sequence:

```
ddg [-{F|f} <config file>] [-{L|l} <log file>]  
    [-{S|s} <SMI object>] [-{T|t} <time-out>] [-{N|n}]  
    [-v] [-h]
```

where:

<code><config file></code>	name of the configuration file. The default is <code>ddg.conf</code> . The configuration file contains all parameters describing the events to be generated.
<code><log file></code>	the name of the log file. The default is the <code>stdout</code> stream.
<code><SMI object></code>	the name of the associated SMI object to connect to. In the form <code><domain_name>::<object_name></code> . The default is: <code>DDG::DDG</code> .
<code><time-out></code>	time-out value for DDL commands. The default is : 1000 μ sec.
<code>-n</code>	do not scatter data (use only 1 data buffer).
<code>-v</code>	verbose mode (capital V is not accepted).
<code>-h</code>	prints a short help and stops.

The aim of the program:

The program is designed for supplying data (simulated events) for the DDL Data Generator card or for the D-RORC card. The program can handle more than one DDL channel (maximum 12 channels). The program works with integrated D-RORC cards, i.e. one RORC card containing 2 DDL channels.

The program reads sub-events from files or generates them according to the user request. The program can also generate DDL header for each sub-events. If header is requested the Event ID of all sub events will be synchronized.

Several replica of the `ddg` program can run parallel on the same or different machines. With carefully set run parameters the DDL headers of the corresponding sub-events (generated with different program replica) remain synchronized.

The simulated sub-event data blocks to be sent to the DDG has to be written into files beforehand. It can happen that the same file has to be used for several channels. In this case one has to make several copies of the file.

The program works in the following way:

- First the program reads the configuration file. Initializes the *physmem* memory, the DIM and SMI packages. The program sets the SMI IDLE state. Opens the requested DDL channels. Opens the data files and reads in the first sub-events into his buffer. Generate the sub-events if they are not read from file.
- The program waits for the READY TO RECEIVE commands from the DDL channels. It should receive them when the DATE system at the receiving ends have been started and are ready to receive data. The program resets the given channel if requested.
- The program should start sending data when a START message from the Experimental Control System had been received. It sets the SMI RUNNING state. It should stop sending data when receiving a STOP message.
- During data sending the program fills sub-event parameters (physical address and length) into the transmit FIFOs of the RORC cards. It checks the status of the FIFOs. When a sub-event is transmitted the program reads in the next from file and pushes its parameters.
- After receiving the STOP message, the program stops sending data and terminates.

Syntax of the data files:

- Any number of empty lines is allowed.
- Lines starting with a "#", "*" or ";" character are considered as comment.
- After ";" or "/" characters the remaining part of any line is considered as in-line comment.

The structure of the data file:

```
<Maximum length in 32 bit words of the sub-events> // decimal
# It must be greater than 0 and less than 16777215 (= 2^24 - 1 = 16MWord - 1)
```

```
<Length in 32 bit words of the first sub-event> // hexa
# It can be 0 and must be less than 0xFFFFFFFF (= 2^24 - 1 = 16MWord - 1)
<Word1> <word2> // hexa
```

```
<Length in 32 bit words of the second sub-event>
<Word1> <word2> // hexa
```

- Space, tabulator or new line characters(s) can separate the above data words.
- The maximum length and the lengths of sub-events must be written in separate lines.

Instruction for ddg program

Syntax of the configuration file:

- The key words and parameters can be separated by space(s), tabulator(s) or equal sign(s).
- Each key word should be written in a separate line.
- Any number of empty lines is allowed.
- Lines starting with a "#", "*" or ";" character are considered as comment.
- After ";" or "///" characters the remaining part of any line is considered as in-line comment.

The use of the key words is not obligatory. Each key word has a default value, which will be used when the key word is not given in the configuration file. In an extreme case the configuration file can be empty.

If the same key word occurs more than once in the configuration file then the last value will be used. This rule applies for contradictory key words as well. For example if one key word specifies to use random event lengths and later on another one specifies no random lengths then the last one will be considered.

The use of following key words are possible:

1. / Channel independent key words:

These key words can be given once for each program instances.

PHYSMEM_OFFSET <offset in MB>

Where

<offset in MB> specifies from which relative address the program can use the *physmem* memory. The use of this and the following parameters makes possible that several instances of ddg program can use different parts of the *physmem* in parallel.

Data format: integer number

Data unit: megabyte

Default value: 0

PHYSMEM_LENGTH <size in MB>

Where

<offset in MB> specifies what part of the *physmem* memory the program can use. The use of this and the preceding parameters makes possible that several instances of ddg program can use different parts of the *physmem* in parallel.

Data format: integer number

Data unit: megabyte

Default value: 32

DDL_COMMANDS

If this keyword is given then the program waits for the RDYRX commands before starting sending data.

Default: if neither DDL_COMMANDS nor NO_DDL_COMMANDS is given then NO_DDL_COMMANDS is supposed.

NO_DDL_COMMANDS

If this keyword is given then the program does not wait for the RDYRX commands before starting sending data.

Default: if neither DDL_COMMANDS nor NO_DDL_COMMANDS is given then NO_DDL_COMMANDS is supposed.

HEADER

If this key word is given then the program generates DDL headers for each sub-event.

Default: if neither HEADER nor NOHEADER is given then NOHEADER is supposed.

NOHEADER

If this key word is given then the program does not generate DDL headers.

Default: if neither HEADER nor NOHEADER is given then NOHEADER is supposed.

BUNCH_CROSSING_START <start value>

Where

<start value>

This value is used to calculate the start values of *Bunch Crossing* and *Orbit Number*, to be used in the first sub-event header of all channels. The following algorithm is used:

```
orbit_start = <start value> / 3564;  
bunch_crossing_start = <start value> % 3564;
```

Data format: integer number

Default value: 1

Instruction for ddg program

BUNCH_CROSSING_INCREMENT <min value> <max value>

Where

<min value>

<max value>

These values specify the minimum and maximum exponents used for incrementing the Bunch Crossing value. The `bcIncrMin` and `bcIncrMax` values, used in the calculation of the new *Bunch Crossing* and *Orbit Number* values (see at BUNCH_CROSSING_SEED keyword) are calculated in the following way:

```
bcIncrMin = 2^<min value> - 1;
```

```
bcIncrMax = 2^<max value> - 1;
```

Data format: integer numbers

Data range: [0 – 31],

<min value> should be less or equal to <max value>

Default values: <min value> = 1, <max value> = 20

BUNCH_CROSSING_SEED <seed value>

Where

<seed value>

This value will be used starting random number generation for calculating the increment of Bunch Crossing. The new *Bunch Crossing* and *Orbit Number* values are calculated using the following algorithm:

```
// random increment between bcIncrMin and bcIncrMax
bcIncrement = bcIncrMin + (int)((float)random() *
    (float)(bcIncrMax - bcIncrMin) / (RAND_MAX+1.0));
orbitNumber += bcIncrement / 3564;
bunchCrossing += bcIncrement % 3564;
if (bunchCrossing >= 3564)
{
    orbitNumber++;
    bunchCrossing %= 3564;
}
```

Data format: integer number

Default value: 0

MAX_EVENT <maximal event number>

Where

<maximal event number>

The number of events to send. If this number is reached in the first channel, the program stops. Use this option only for program tests.

Data format: integer number

Default: 0: the program stops only if SMI command STOP is received.

2. / Channel dependent key words:

These key words can be given for each DDL channels.

RORC_CHANNEL <minor> <channel>

Where

<minor>

Defines the minor number of the D-RORC device. Using the `rorc_find` DDL program one can find the minor numbers of the device.

Data format: integer number

Default value: 0

<channel>

Defines the channel number of the D-RORC card. It can be 0 (channel A) or 1 (Channel B).

Data format: integer number

Default value: 0

If DDL channel is not specified then only one channel (minor = 0, channel = 0) will be supposed.

DATA_FILE <file name>

Where

<file name>

The name of the data file containing the simulated sub-events for the given channel.

Data format: string of maximum 80-character length.

Default value: `ddg.data`

If both `DATA_FILE` and `DATA_PATTERN` are given, the last one will be considered. If neither is given then `DATA_PATTERN` will be supposed.

DATA_PATTERN <pattern>

Where

<pattern>

If this key word is given the program generates the simulated data for the given channel. One can choose from the following possibilities:

'c' :	constant data
'a' :	alternating data
'0' :	“flying” binary 0 pattern
'1' :	“flying” binary 1 pattern
'i' :	incrementing data

Instruction for ddg program

'd' : decrementing data

The first data word of the generated sub-events will be the event's serial number (starting from 1). The second word will be the init word defined by the INIT_WORD keyword, and then follows the data words as specified by the <pattern>.

Data format: character

Default value: i

If both DATA_FILE and DATA_PATTERN are given, the last one will be considered. If neither is given then DATA_PATTERN will be supposed.

INIT_WORD <start value>

Where

<start value>

The value of the second word of the generated sub-events for the given channel.

Data format: hexadecimal number

Default value: 0xffffffffe for "flying" 0 pattern,
 0x00000001 for "flying" 1 pattern,
 0 for other patterns

DATA_LENGTH <maximum length>

Where

<maximum length>

The length of generated sub-event patterns in 32-bit words. If RANDOM is not specified this will be the size of the sub-event blocks. If RANDOM is specified then this will be the maximum size of the blocks.

Data format: integer number

Data range: [1 – 16777215 (= 2²⁴ – 1 = 16MWord - 1)]

Default value: 524287 (= 512kWord – 1)

RANDOM

If this key word is given then the program generates random length sub-events. The minimum length is 0. The key word DATA_LENGTH or the length of sub-event read from data file specifies the maximum random length.

Default: if neither RANDOM nor NORANDOM is given then RANDOM is supposed.

NORANDOM

If this key word is given then the program generates sub-events with lengths specified by the DATA_LENGTH key word or using the length of the sub-event read from the data file.

Default: if neither RANDOM nor NORANDOM is given then RANDOM is supposed.

RESET

If this key word is given the program resets the given DDL channel before starting to generate the sub events.

Default: if neither RESET nor NORESET is given then RESET is supposed.

NORESET

If this key word is given the program does not reset the given DDL channel.

Default: if neither RESET nor NORESET is given then RESET is supposed.

3. / Key words describing DDL headers:

These key words can specify the DDL header for each channel. For the detailed description of header fields see note:

ALICE-INT-2002-10 v.5: Data Format over the ALICE DDL,
https://edms.cern.ch/cedar/plsql/doc.info?cookie=3148766&document_id=340186&version=5

BLOCK_LENGTH

If this key word is given then the program fills the *block length* field of each DDL header of the given channel with the lengths of the sub events.

Default: if neither BLOCK_LENGTH nor NO_BLOCK_LENGTH is given then BLOCK_LENGTH is supposed.

NO_BLOCK_LENGTH

If this key word is given then the program fills the *block length* field of each DDL header of the given channel with 0xFFFFFFFF.

Default: if neither BLOCK_LENGTH nor NO_BLOCK_LENGTH is given then BLOCK_LENGTH is supposed.

MINI_EVENT_ID

If this key word is given then the program puts into the *mini-event ID* field the value of the *bunch crossing*.

Default: if neither MINI_EVENT_ID nor NO_MINI_EVENT_ID is given then MINI_EVENT_ID is supposed.

Instruction for ddg program

NO_MINI_EVENT_ID

If this key word is given then the program generates random errors in the *mini-event ID* header field.

Default: if neither MINI_EVENT_ID nor NO_MINI_EVENT_ID is given then MINI_EVENT_ID is supposed.

FORMAT_VERSION <version>

Where

<version>

The format version of the DDL header.

Data format: hexadecimal number

Data size: 1 byte

Default value: 1

L1_TRIGGER <L1 trigger message>

Where

<L1 trigger message>

Simulated L1 trigger message.

Data format: hexadecimal number

Data size: 10 bits

Default value: 0

SUB_DETECTORS <participating sub-detectors>

Where

<participating sub-detectors>

Data format: hexadecimal number

Data size: 3 bytes

Default value: 0

ATTRIBUTES <block attributes>

Where

<block attributes>

Data format: hexadecimal number

Data size: 1 byte

Default value: 0

STATUS_BITS <status and error bits>

Where

<status and error bits>

Data format: hexadecimal number

Data size: 2 bytes

Default value: 0

TRIGGER_CLASS_LOW <trigger class low bits>

Where

<trigger class low bits>

The low part (bits 1-31) of the trigger class information

Data format: hexadecimal number

Data size: 4 bytes

Default value: 0

TRIGGER_CLASS_HIGH <trigger class high bits>

Where

<trigger class low bits>

The high part (bits 32-49) of the trigger class information.

Data format: hexadecimal number

Data size: 18 bits

Default value: 0

ROI_LOW <ROI low bits>

Where

<ROI low bits>

The low part (bits 0-3) of the Region of Interest information

Data format: hexadecimal number

Data size: 4 bits

Default value: 0

ROI_HIGH <ROI high bits>

Where

<ROI high bits>

The high part (bits 4-35) of the Region of Interest information

Data format: hexadecimal number

Data size: 4 bytes

Default value: 0