Data Request Format 2.0

Request for Comments • Revision 2 • September 2, 2009

K. Cahill, B. Hendricks, C. King, R. Neswold, J. Patrick, A. Petrov, and C. Schumann

Edited by Andrey Petrov <apetrov@fnal.gov>

Last version of this document: http://www-bd.fnal.gov/controls/public/drf2

Revision History

Table Of Contents

1. Introduction

This document describes a uniform data request format for the use in applications, data acquisition protocols, and the middleware (DAE, DPM) of the Fermilab Accelerator Control System.

The established practice of denominating control system's entities—devices, properties, and events—originates from several protocols and APIs developed long time ago. Intuitive, yet somewhat ambiguous, the naming conventions have never been fully documented independently of their implementations. As such, various pieces of software interpret those concepts differently, making their own assumptions about the permitted syntax and character sets. The further evolution of the control system calls for a review of the existing naming practices, in order to make them more rational, remove ambiguity, and to formalize the syntax.

The *Data Request Format, Version 2* (DRF2) is based on the existing naming conventions. It is backward compatible, meaning that the new format will honor currently accepted device names, property qualifiers, array indices and range denominators, event names, as well as relevant ACL property and field names. The new format, however, may not be understood by the existing software, and it is intended only for new implementations of data acquisition clients, middleware, and such. A standard DRF2 parser, fully compatible with this document, is provided in both C++ and Java.

The principal features of DRF2 are:

- The full data request includes five attributes: *device*, *property*, *range*, *field*, and *event*. This precisely describes a quantum of data in the control system that a client wants to read or set. Normally, they are (or directly translated to) the requisites of a single data pool entry (perhaps, excluding the *field* attribute). Note that the reference to a model is not included in the set of attributes, which makes the data request invariant to a redirection.
- The format is lenient and requires only a device name to be specified. One data can be requested in several ways.
- There is one canonical form for each semantically distinct request.
- Within the standard parser, the syntactic validation of a request and its conversion to the canonical notation is performed off-line without consulting external resources.
- The canonical form of a data request can be used as a unique key for a data pool entry. Note however, that due to the complexity of the underlying control system it is impossible to tell, without checking the database, whether two semantically different requests are, in fact, distinct.

This document uses a BNF-like grammar (similar to one from RFC2396) to describe the syntax of DRF2. The following definitions are common to many elements:

```
upalpha         = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
                | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T"
                | "U" | "V" | "W" | "X" | "Y" | "Z"

lowalpha        = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
                | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t"
                | "u" | "v" | "w" | "x" | "y" | "z"

alpha           = upalpha | lowalpha

dec-digit       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

dec-number      = 1*( dec-digit )

hex-digit       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
                | "A" | "B" | "C" | "D" | "E" | "F"
                | "a" | "b" | "c" | "d" | "e" | "f"

hex-number      = 1*( hex-digit )

; Canonical forms

c-dec-number    = <A dec-number with no leading zeroes>

c-hex-number    = <A hex-number with no leading zeroes and all chars capitalized>
```

2. The Request Structure

The data request is defined as a case-insensitive text string with all characters in the ASCII range 0x21 (“!”) to 0x7E (“~”).

The request includes an ordered sequence of up to five attributes: *device*, *property*, *range*, *field*, and *event*. The only required attribute is *device*.

```
request         = device [ "." property ] [ range ] [ "." field ] [ "@" event ]
```

In the canonical form of a request each attribute is individually transformed to its canonical notation. A missing *property* attribute is replaced with its default value, as described in §4. The *range*, *field*, and *event* attributes are present in the canonical form only if their values are not default. The notion of a *NULL* attribute used later in this document describes the entire attribute absent from the request along with its delimiter, not the delimiter and an

empty string.

3. Device Format

The *device* attribute specifies a long ACNET device name. For devices existed before the introduction of long names, a long name is the same to a short 8-character name. A long device name contains 3 to 64 characters and begins with a letter. The second character may designate a default property.

```
device          = dev-pref prop-qualifier 1*62( dev-char )

dev-pref        = alpha

dev-char        = alpha | dec-digit | "_" | ":"

prop-qualifier  = ":"       ; Reading and canonical
                | ";"       ; Reading, Java style
                | "?"       ; Reading, ACL style
                | "_"       ; Setting
                | "|"       ; Basic Status
                | "&"       ; Basic Control
                | "@"       ; Analog Alarm
                | "$"       ; Digital Alarm
                | "~"       ; Description
```

In the canonical form the property qualifier is changed to a colon. Whenever possible, the original character case shall be preserved to facititate recognition of device names by the users.

```
; Canonical

c-device        = dev-pref ":" 1*62( dev-char )
```

4. Property Format

The *property* attribute specifies the name of a property within the device. The format supports seven properties: Reading, Setting, Basic Status, Basic Control, Analog Alarm, Digital Alarm, and Description. This list may be extended in the future. Each property has one canonical name and several synonyms.

The default property is specified by a property qualifier inside the device name. If a property is explicitly specified in a request, the property qualifier inside the device name must either match that property or be a colon.

```
; Canonical property names are in the upper case

property        = reading
                | setting
                | basic-status
                | basic-control
                | analog-alarm
                | digital-alarm
                | description

reading         = "READING"                                 ; Canonical
                | "READ" | "PRREAD"

setting         = "SETTING"                                 ; Canonical
                | "SET" | "PRSET"

basic-status    = "STATUS"                                  ; Canonical
                | "BASIC_STATUS" | "STS" | "PRBSTS"

basic-control   = "CONTROL"                                 ; Canonical
                | "BASIC_CONTROL" | "CTRL" | "PRBCTL"
```

```
analog-alarm    = "ANALOG"                                  ; Canonical
                | "ANALOG_ALARM" | "AA" | "PRANAB"

digital-alarm   = "DIGITAL"                                 ; Canonical
                | "DIGITAL_ALARM" | "DA" | "PRDABL"

description     = "DESCRIPTION"                             ; Canonical
                | "DESC" | "PRDESC"
```

5. Range Format

The *range* attribute addresses an individual element or a sequence of elements in the data set.

A data set can be viewed either as an array of homogenous components or as a sequence of raw bytes. Accordingly, a range may be specified in two forms, depending on which model is used. Array indices are enclosed in brackets, a byte offset and a length are enclosed in braces. One shall make no assumption, based on a form of the range, whether the request asks for raw or scaled data. The bytewise form is deprecated, because it requires the users to know low-level details of data representation and is prone to scaling errors.

```
range           = full-range
                | array-range
                | byte-range
                ; Default is "[0]"

full-range      = "[]" | "[:]" | "[0:]"
                | "{}" | "{:}" | "{0:}"

array-range     = ( "[" start-index              "]" )      ; end-index = start-index
                | ( "[" start-index ":"          "]" )      ; up to the end
                | ( "["             ":" end-index "]" )      ; start-index = 0
                | ( "[" start-index ":" end-index "]" )
                ; 0 ≤ start-index ≤ end-index < 2^31

byte-range      = ( "{" offset           "}" )               ; length = 1
                | ( "{" offset ":"       "}" )               ; up to the end
                | ( "{"        ":" length "}" )               ; offset = 0
                | ( "{" offset ":" length "}" )
                ; 0 ≤ offset < 2^31
                ; length > 0
                ; offset + length ≤ 2^31

start-index     = dec-number

end-index       = dec-number

offset          = dec-number

length          = dec-number
```

The canonical form of a range is defined as follows:

```
; Canonical

c-range         = c-full-range
                | c-array-range
                | c-byte-range

c-full-range    = "[]"

c-array-range   = NULL                                      ; =[0]
                  ( "[" c-start-index                "]" ) ; c-start-index > 0
                | ( "[" c-start-index ":"            "]" ) ; c-start-index ≥ 0
                | ( "[" c-start-index ":" c-end-index "]" ) ; 0 ≤ c-start-index < c-end-index

c-array-range   = ( "{" c-offset             "}" )          ; c-offset ≥ 0
                | ( "{" c-offset ":"         "}" )          ; c-offset ≥ 0
```

```
                | ( "{" c-offset ":" c-length "}" )         ; c-length > 1

c-start-index   = c-dec-number

c-end-index     = c-dec-number

c-offset        = c-dec-number

c-length        = c-dec-number
```

6. Field Format

The *field* attribute specifies a flavor of data, such as raw, scaled, or a particular field inside a complex structure. Each property has its own set of valid fields.

Fields can be static or dynamic. All static fields are pre-defined, so that the parser can process them off-line (e.g., change field names upon the convertion to the canonical form). The set of possible dynamic fields may depend on the device and is unknown to the parser. The validity of dynamic fields can not be verified.

```
; Canonical field names are in the upper case.

field           = reading-fld
                | setting-fld
                | status-fld
                | control-fld
                | analog-fld
                | digital-fld
                | description-fld

reading-fld     = "RAW"
                | "PRIMARY" | "VOLTS"                     ; Canonical is "PRIMARY"
                | "SCALED" | "COMMON"                      ; Default

setting-fld     = "RAW"
                | "PRIMARY" | "VOLTS"                     ; Canonical is "PRIMARY"
                | "SCALED" | "COMMON"                      ; Default

status-fld      = "RAW"
                | "SCALED" | "COMMON"                      ; Default
                | dynamic-fld

control-fld     = NULL                                     ; Default

analog-fld      = NULL                                     ; Default
                | "MIN" | "MINIMUM"                        ; Canonical is "MIN"
                | "MAX" | "MAXIMUM"                        ; Canonical is "MAX"
                | "NOM" | "NOMINAL" | "ANALOG_NOMINAL"     ; Canonical is "NOM"
                | "TOL" | "TOLERANCE"                      ; Canonical is "TOL"
                | "ALARM_ENABLE"
                | "ALARM_STATUS"
                | "TRIES_NEEDED"
                | "TRIES_NOW"
                | "ALARM_FTD"
                | "ABORT"
                | "ABORT_INHIBIT"
                | "LIMIT_TYPE"
                | "FLAGS"
                | "RAW"

digital-fld     = NULL                                     ; Default
                | "NOM" | "NOMINAL"                        ; Canonical is "NOM"
                | "MASK"
                | "ALARM_ENABLE"
                | "ALARM_STATUS"
                | "TRIES_NEEDED"
                | "TRIES_NOW"
                | "ALARM_FTD"
                | "ABORT"
                | "ABORT_INHIBIT"
                | "FLAGS"
```

```
                | "RAW"

description-fld = NULL                                      ; Default

dynamic-fld     = dynamic-pref 0*63( dynamic-char )

dynamic-pref    = alpha | "_"

dynamic-char    = alpha | dec-digit | "_"

; Canonical

c-dynamic-fld   = c-dynamic-pref 0*63( c-dynamic-char )

c-dynamic-pref  = c-alpha | "_"

c-dynamic-char  = c-alpha | dec-digit | "_"
```

7. Event Format

The *event* attribute specifies on which moments of time the data should be read or set. The syntax of event descriptors is similar to one used in the Get32 protocol and in the DataEventFactory class.

```
event           = default-evt
                | immediate-evt
                | periodic-evt
                | clock-evt
                | state-evt
                ; Default is "U"

default-evt     = "U"

immediate-evt   = "I"

periodic-evt    = "P" [ "," period [ "," flag ] ]
                ; Default period is "1000"
                ; Default flag is "TRUE"
                ; 0 < period < 2^31

period          = dec-number                                ; Period in milliseconds

flag            = "TRUE" | "FALSE" | "T" | "F"              ; Immediate notification flag

clock-evt       = "E," evt-number [ "," type [ "," delay ] ]
                ; Default type is "E"
                ; Default delay is "0"
                ; 0 ≤ evt-number < 2 ^16
                ; 0 ≤ delay < 2 ^31

evt-number      = hex-number                                ; Event number

type            = "H" | "S" | "E"                           ; Hardware / Software / Either One

delay           = dec-number                                ; Delay in milliseconds after an S event

state-evt       = "S," source "," value "," delay "," expression
                ; 0 ≤ value < 2 ^16

source          = c-device | device-index
                ; 0 ≤ device-index < 2 ^31

device-index    = dec-number

value           = dec-number

expression      = "=" | "!=" | ">" | "<" | "<=" | ">=" | "*"
```

The canonical form of an event is defined as follows:

```
;Canonical

c-event         = c-default-evt
                | c-immediate-evt
                | c-periodic-evt
                | c-clock-evt
                | c-state-evt

c-default-evt   = NULL

c-immediate-evt = "I"

c-periodic-evt  = "P," c-period "," c-flag

c-period        = c-dec-number

c-flag          = "TRUE" | "FALSE"

c-clock-evt     = "E," c-evt-number "," c-type "," c-delay

c-evt-number    = c-hex-number

c-type          = "H" | "S" | "E"

c-delay         = c-dec-number

c-state-evt     = "S," c-source "," c-value "," c-delay "," c-expression

c-source        = c-device | c-device-index

c-device-index  = c-dec-number

c-value         = c-dec-number

c-expression    = "=" | "!=" | ">" | "<" | "<=" | ">=" | "*"
```

8. Test Vectors

Valid requests:

Input	Canonical Form
m:outtmp	m:outtmp.READING
C:A7CVPF.SET	C:A7CVPF.SETTING
I~BEAM	I:BEAM.DESCRIPTION
Z@Void[0]	Z:Void.ANALOG
u@cpsts.alarm_enable@p,0333,f	u:cpsts.ANALOG.ALARM_ENABLE@P,333,FALSE
G_WYFI2M{}.RAW@p	G:WYFI2M.SETTING[].RAW@P,1000,TRUE
"L:W5H{0:}.RAW	L:W5H.READING[].RAW
T:baigsd.prdabl[:7]@e,2a	T:baigsd.DIGITAL[0:7]@E,2A,E,0
C_BAVGX.VOLTS	C:BAVGX.SETTING.PRIMARY
c|boq2f.dynamic_foo@s,V:Test,233,0,*	c:boq2f.STATUS.DYNAMIC_FOO@S,V:Test,233,0,*
c?frau1dh{:4}.raw@i	c:frau1dh.READING{0:4}.RAW@I
d:PD_KLY1:Kly:Cplr:FwdPwr.aa.nom@u	d:PD_KLY1:Kly:Cplr:FwdPwr.ANALOG.NOM

Invalid requests:

Input	Error
T:QX Ing	Illegal whitespace
t:sq+z0	Illegal "+" character

`S^SYPI1`	Illegal property qualifier
`z_cache.read`	The property qualifier doesn't match the property attribute
`c:frau1dh.raw{:4}@i`	The range attribute is misplaced
`c:void.rd`	Illegal property name
`b:trzdk.setting.bytes`	Illegal field name
`m:blow.scaled.reading`	Illegal order of attributes
`L:TR0KK[7:1]`	Illegal end index
`M:ZZ0@S,V:TEST`	Incomplete state event

$Id: drf2.html,v 1.21 2009/09/02 16:05:02 apetrov Exp $

Security, Privacy, Legal