Piezo · Nano · Positioning

GCS Array Data Format Description

Release: 1.2.1 Date: 2010-09-27

The described data format is that used by the PI General Command Set (GCS).

Moving the NanoWorld | www.pi.ws

Table of Contents

0. Manufacturer Declarations 2
0.1. Disclaimer 2
1. Introduction 3
2. Conventions 3
3. Header 4
3.1. Description 4
3.2. Keywords 5
4. Values and units 6
5. Data 7
5.1. Matrix Data 7
5.1.1. Description 7
5.1.2. Order of Data 9
5.2. Table Data 10
5.2.1. Description 10
5.2.2. Time Series 10
5.2.3. Order of Data 11

Release: 1.2.1
File:GCSData_User_SM146E.doc, 207872 Bytes

0. Manufacturer Declarations

The contents of the manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Physik Instrumente (PI).

Physik Instrumente (PI) assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

0.1. Disclaimer

Physik Instrumente (PI) does not guarantee that this data format description is free of errors and will not be responsible for any damage arising from its use.

Physik Instrumente (PI) expressly disclaims any and all warranties including any implied warranty of merchantability and fitness for a particular purpose. Physik Instrumente (PI) does not warrant, guarantee, or make any obligations regarding the use or the results of the use of this data format, in terms of correctness, accuracy, reliability or otherwise. The user agrees to use this data format on his own responsibility.

1. Introduction

The PI General Command Set (GCS) is common to most PI piezo and motor controllers, and is designed for multi-axis operation. This greatly reduces the effort required to produce custom programs, especially in environments which include a number of different controllers. PI offers GCS-compatible user-interface software as well as LabView™ and other driver sets.

The *GCS Array* data format was defined to enable the exchange of complex data between controllers and/or user interface software. The format provides for transfer of n-dimensional data arrays as well as meta-information about the data transmitted (for example the name of the data source and, optionally, user-provided or application-specific remarks).

2. Conventions

The *GCS Array* data specification defines the conversion of an n-dimensional array of numeric values to an ASCII stream called a *dataset*. One or more *datasets* can be stored in a file and/or transmitted over an interface such as RS-232 or GPIB. Any extension may be used in *GCS Array* filenames, except extensions reserved for other uses. See the manuals for the appropriate controller and software for detailed information on how to save data in *GCS Array* format.

A *dataset* always consists of a *header* followed by data and is almost self-explanatory, since the *GCS Array* data format is based on plain ASCII. The first character of each line belonging to the header is "#". The first line not starting with "#" marks the end of the header and is the first line of the data section. For a detailed description of header and data see sections 3 and 4.

If a file includes more than one *dataset*, the individual datasets are identified by their *Names* and separated by lines containing "[GCS_ARRAY *dataset_name*]". You can type in the entry for *Name* during the saving procedure (see the appropriate controller and software manuals for details). The name may not contain the "[", "]" or ";" characters.

Example:

```
[GCS_ARRAY BC-Scan]
# REM data scanned with C-880 Control
#
# TYPE = 0
# SEPARATOR = 9
# DIM = 3
#
# START0 = 0.3
# END0 = 0.6
# NDATA0 = 13
# NAME0 = B [mm]
#
# START1 = 0.3
# END1 = 0.6
# NDATA1 = 4
# NAME1 = C [mm]
#
# NDATA2 = 52
# NAME2 = Intensity [V]
0.00198	0.00107	0.00153	0.00153
0.00183	0.00153	0.00198	0.00168
```

```
0.00153     0.00168      0.00107      0.00183
0.00122     0.00122      0.00198      0.00153
0           5.80621      0            0
0.00137     0.00137      0.00168      0.00122
0.00183     0.00153      0.00198      0.00122
0.00107     0.00122      0.00122      0.00153
0.00168     0.00198      0.00137      0.00137
0.00153     0.00198      0.00107      0.00122
0.00137     0.00168      0.00183      0.00168
0.00137     0.00153      0.00183      0.00198
0.00107     0.00153      0.00122      0.00198
[GCS_ARRAY XY-Scan]
# REM intensity along a path in X and Y
#
# TYPE = 1
# SEPARATOR = 9
# DIM = 3
# NDATA = 5
#
# NAME0 = X position [mm]
# NAME1 = Y position [mm]
# NAME2 = intensity [V]
#
2.1         -4.02      0.001
2.24        -6.93      0.0021
2.4         -8.01      0.0019
2.524       -7.03      0.00562
2.802       0          0.00341
```

Listing 1: Sample file containing two datasets (“BC-Scan” and “XY-Scan”) in GCS Array data format

3. Header

3.1. Description

A dataset starts with the header. The first character of each line belonging to the header is “#”. The first line not starting with “#” marks the end of the header and is the first line of the data. Any header line found in the dataset after this point will cause an error when read in.

The keyword VERSION indicates the version of the *GCS Array* data format. (In the future new versions may be introduced). If VERSION is not specified in the header, Version 1 is assumed.

Each header line can hold a single value, or a remark, or can be empty. To identify the values and the remarks, each header line starts with a *keyword*. Recognition of keywords is case insensitive (e.g. REM, rem and REm are all interpreted as REM).

The header can contain remarks whose contents are not part of the *GCS Array* data specification. Information like the date of creation or the author of the data can be placed in remarks. A remark starts with the keyword REM.

Lines holding values have the following syntax:

# KEY = value

where *KEY* stands for one of the defined keywords, and *value* is a numeric value, either integer or floating point.

Valid keywords are listed in the table below. Note that the percent sign (%) is to be replaced by a non-negative integer.

3.2. Keywords

Keyword	**Description**	**For Data Type**	**Required / Optional (with default value)**
REM	Everything in the same line after “REM“ is ignored	All	optional
VERSION	Version of *GCS Array* format used	All	Required for version >1
DIM	Number of “dimensions” of data; minimum 2; use 3 for 2-D scan + intensity data	All	Required
TYPE	Type of data: currently there are two data types: • 0: matrix data (see p. 7) • 1: table data (see p. 10)	All	Required
SEPARATOR	When the data sections contains more than one data element in a single line, the elements are separated by the “separator” character. This value is the decimal ASCII value of the separator character (e.g. a value of 9 is the TAB character, 32 is the space character, ...)	All	Optional, default is 9 (TAB), 32 (SPACE) is always treated as separator
NDATA	Number of rows in the table (number of n-dimensional data points)	Table	Required
NDATA% e.g.: NDATA0 NDATA1 ...	Size of the % dimension, where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3).	Matrix	Required for all but the “highest” dimension
NAME%	Value which can be used to describe the data in the % dimension (e.g. you can use this as the caption for the corresponding axis in a graph), where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3).	All	Optional
START%	Start value for the % dimension, where % is a non-negative integer smaller than DIM–1 (for DIM=4, % goes from 0 to 2). See documentation of matrix data for details (p. 7).	Matrix	Required
END%	End value for the % dimension, where % is a non-negative integer smaller than DIM–1 (for DIM=4, % goes from 0 to 2). See documentation of matrix data for details (p. 7).	Matrix	Required, if “DELTA%” is not given
DELTA%	Difference between values for the % dimension, where % is a non-negative integer smaller than DIM–1 (for DIM=4, % goes from 0 to 2).	Matrix	Required, if “END%” is not given
SAMPLE_TIME	If the table data is a time series, this value gives the time difference between two data points	Table	Optional

Keyword	Description	For Data Type	Required / Optional (with default value)
TRANS_UNIT%	Unit of transmitted values as text, where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3). If this is “RAW” DISP_UNIT%, RATIO_NOM% and RATIO_DENOM% can be used to calculate and display some more readable form	All	Optional
DISP_UNIT%	Unit of displayed values as text, where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3). If TRANS_UNIT is “RAW”, RATIO_NOM% and RATIO_DENOM% can be used to calculate and display some more readable form	All	Optional
RATIO_NOM%	Nominator of factor to transform transmitted value, where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3).	All	Optional
RATIO_DENOM%	Denominator of factor to transform transmitted value, where % is a non-negative integer smaller than DIM (for DIM=4, % goes from 0 to 3).	All	Optional
END_HEADER	Marks the last line of the header	All	Required

4. Values and units

To achieve more accuracy with small values and to save CPU time on the controller side it may be better e.g. to send integer data of raw encoder counts and let the receiving host software convert these values to “real world” data (e.g. mm). To indicate the format of transmitted values TRANS_UNIT can be used. If TRANS_UNIT is “RAW” the host software can look up DISP_UNIT, RATIO_NOM and RATIO_DENOM to convert the data read.

Example:

```
[GCS_ARRAY BC-Scan]
# …
#
# TRANS_UNIT0 = RAW
# TRANS_UNIT1 = RAW
# TRANS_UNIT2 = V
# DISP_UNIT0 = mm
# DISP_UNIT1 = mm / sec
# RATIO_NOM0 = 1
# RATIO_DENOM0 = 1000
# RATIO_NOM1 = 1
# RATIO_DENOM1 = 4000
#
# …
```

In this example the position is transmitted as counts and the velocity as counts/cycle. To display the values as mm or as mm/sec the values read in must be converted:

$$position[mm] = \frac{RATIO\_NOM0}{RATIO\_DENOM0} \cdot position[raw]$$

$$velocity[mm/\sec] = \frac{RATIO\_NOM1}{RATIO\_DENOM1} \cdot velocity[raw]$$

The values of the third channel are transmitted already in V (volts), so no conversion is necessary.

5. Data

Version 1 of the *GCS Array* data format recognizes two types of n-dimensional arrays of values, known as *matrix* data (TYPE = 0) and *table* data (TYPE = 1).

An array can be handled as matrix data if the values of the first n-1 dimensions are equally spaced, i.e. fully determined by the *start value*, *end value* and the *number of values* value for the dimension. When *matrix data* is specified in the TYPE keyword, the data section contains only values from the nth dimension: the values for the other dimensions can be regenerated from their respective *start, end* and *number* values.

If the array does not have the regularity required for handling as *matrix data*, it can be handled as *table data*. When *table data* is specified in the TYPE keyword, the data section contains the values of all n dimensions for each data point.

Matrix data can be transferred more rapidly than table data.

5.1. Matrix Data

5.1.1. Description

Data TYPE 0, *matrix data,* is used for “uniform” data, i.e. data where the first n-1 dimensions contain values which are equally spaced. Such datasets could result, for example, when data is collected from a scan along one or more axes with all steps on a given axis the same size.

Example 1 (see *Listing 2* below): While moving an axis in 0.1 mm steps, the intensity of the analog input channel is measured. In this case you do not need to store the position for each of the data points. It is sufficient to place the start position, the end position and the number of steps in the header in order to know everything relevant about the motion of the axis. The only values written to the data section of the dataset are the measured intensity values.

Example 2 (see also *Listing 3* on p. 9): A 2-axis rectangular scan moves over a grid with uniform steps and measures the intensity at each point. The only values which are needed for the first two axes are the start and the end positions and the number of data points along each axis. These values are placed in the header. The data section contains only the measured intensity values.

Hence, for matrix data, the data actually written to the data section of the dataset is always the data for the highest-level “dimension”. The lower-level dimensions are described completely by the START%, END% or DELTA% and NDATA% values in the header.

The following file contains one dataset, which describes a scan along a single axis, named the B-axis, from 0.38 to 0.42, with 9 collected data points.

```
# REM data scanned with C-880 Control
# REM saved 10:04:36 - Tuesday, September 10, 2002
#
# TYPE = 0
# SEPARATOR = 9
# DIM = 2
#
# START0 = 0.38
# END0 = 0.42
# NDATA0 = 9
# NAME0 = B [mm]
#
# NDATA1 = 9
# NAME1 = Intensity [V]
#
0.00137
0.00137
0.00107
2.72282
5.80789
1.18349
0.00183
0.00168
0.00107
```

*Listing 2: Scan along one axis with steps of uniform size; saved as matrix data (*TYPE 0*)*

From the information in the header you can calculate the corresponding B-position for each measured intensity value. B was 0.38 for the first value (0.00137) and 0.395 for the 4th value (2.72282). To calculate the step size between two points, use the following formula:

STEPS% = (END% - START%) / (NDATA%-1)

As an alternative you can have the step size given in the header with the key DELTA%. To calculate the last value of the data use

END% = START% + (NDATA%-1) * DELTA%

Note that START%, END%, DELTA% and NDATA% are required for each dimension but the highest, because without them, the values for these dimensions can not be calculated. For the highest dimension, the values are found in the data section.

The following file contains a dataset which shows a 2D scan using axes designated B and C with 13 data points along the B axis and 4 along the C axis.

```
# REM data scanned with C-880 Control
#
# TYPE = 0
# SEPARATOR = 9
# DIM = 3
#
# START0 = 0.3
# END0 = 0.6
# NDATA0 = 13
# NAME0 = B [mm]
#
# START1 = 0.3
# DELTA1 = 0.1
# NDATA1 = 4
# NAME1 = C [mm]
```

```
#
# NDATA2 = 52
# NAME2 = Intensity [V]
0.00198     0.00107     0.00153     0.00153
0.00183     0.00153     0.00198     0.00168
0.00153     0.00168     0.00107     0.00183
0.00122     0.00122     0.00198     0.00153
0           5.80621     0           0
0.00137     0.00137     0.00168     0.00122
0.00183     0.00153     0.00198     0.00122
0.00107     0.00122     0.00122     0.00153
0.00168     0.00198     0.00137     0.00137
0.00153     0.00198     0.00107     0.00122
0.00137     0.00168     0.00183     0.00168
0.00137     0.00153     0.00183     0.00198
0.00107     0.00153     0.00122     0.00198
```

*Listing 3: Scan along two axes with steps of uniform sizes; saved as matrix data (*TYPE 0*)*

In this example the organization of the data becomes clear. With this data you can build a matrix *m* of size (13,4). The order of the elements in the data section of the dataset is: *m*(0,0), *m*(0,1), *m*(0,2), *m*(0,3), *m*(1,0), *m*(1,1), ... , *m*(12,3).

Listing 3 shows that at the position with the coordinates B = 0.325 and C=0.4, an intensity of 0.00153 was measured, and the maximum value was found at B=0.4 and C=0.4.

5.1.2. Order of Data

Elements in the data section need not be arranged in the row-column format shown in the example in Listing 3. There could be only one value per line—as long as the information in the header is consistent, the “layout” of the data in the data section does not matter.

For matrix data in a dataset with a given DIM specification you can calculate the number of data elements by multiplying the NDATA% values:

```
NumberOfData = 1;
FOR (I=0 TO DIM-2)
  NumberOfData := NumberOfData * NDATA[i];
NEXT;
```

or, in mathematical notation:

NumberOfDataPoints = (NDATA0)(NDATA1)(NDATA2)...(NDATA*n*)

where *n* = DIM-2

The data is stored in a matrix of dimension DIM-1. The values can be read into a corresponding array (e.g. matrix) in your program with code like the following:

```
FOR (I=0 TO NDATA[0]-1)
        FOR(J=0 TO NDATA[1]-1)
                ...
                FOR(P=0 TO NDATA[DIM-2]-1)
                        matrix[I, J, ..., P] =     ReadNextValueFromFile();
                NEXT;
        NEXT;
NEXT;
```

5.2. Table Data

Data TYPE 1, *table data*, can be used to store data that does not have the regularity required for *matrix data* described above.

5.2.1. Description

Suppose, for example, you measure the intensity of an analog input while moving along a path in space. At each measurement point the X, Y, Z coordinates and the intensity value (four “dimensions”) are captured. Because the path may be irregular, it is not possible to generate any of the coordinate values with a stepping algorithm as it was when scanning, and hence all four data values must be accorded places in the data section.

Think of the data as organized with one column for each dimension and one row for each measurement. For table data the START%, END% and NDATA% keywords are not needed. Only one keyword, NDATA, is required; NDATA gives the number of rows (data points). The number of columns is given by the value for DIM.

```
# REM intensity along a path in X and Y
#
# TYPE = 1
# SEPARATOR = 9
# DIM = 3
# NDATA = 5
#
# NAME0 = X position [mm]
# NAME1 = Y position [mm]
# NAME2 = intensity [V]
#
2.1	-4.02	0.001
2.24	-6.93	0.0021
2.4	-8.01	0.0019
2.524	-7.03	0.00562
2.802	0	0.00341
```

*Listing 4: Motion in a plane and corresponding intensity values saved as table data (*TYPE 1*)*

5.2.2. Time Series

Another example for table data are time series. The difference in time between two data points (“rows” in the table) is fixed. In order to transfer this fixed value only once, there’s the special key “SAMPLE_TIME”.

```
# TYPE = 1
# SEPARATOR = 32
# DIM = 2
# NDATA = 7
# SAMPLE_TIME = 0.04
# NAME0 = position
# NAME1 = position error
# END_HEADER
1.01 0.003
1.013 0.0025
1.01 0.002
1.02 0.002
```

```
1.018 0.0021
1.009 0.002
1.0 0.002
```

*Listing 5: Time series of two measured items as table data (*TYPE 1*)*

5.2.3. Order of Data

In the data section of the data set, the elements need not be arranged in the row-column format shown in the example. It is also allowable to place one value per line (i.e. there is no distinction between the separator and linefeed characters). As long as the information in the header is consistent, the “layout” of the data does not matter, only the order.

The data is organized “row” wise: the first value is the first value for the first dimension, the second value is first value for the second dimension (column), until the values for all dimensions have been read in; then follows the second value for the first dimension, etc.

The data can be stored in a 2D matrix with code like the following:

```
FOR (ROW=0 TO NDATA-1)
    FOR(COL=0 TO DIM-1)
        matrix[ROW, COL] = ReadNextValueFromFile();
    NEXT;
NEXT;
```