metric: Metric Library Reference

CML00006-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Library Interface

1.1 Introduction

This document describes the interface to the Code Magus Limited `metric` library. The library implements an abstract data type describing a metric. A single metric may be serialised into a buffer for consistent delivery to other applications (often across a network or via a file) or de-serialised from a buffer back into the data structure for consumption by an application. Functions are also provided for creating, updating and sending a metric to a server.

2 API Reference

The API interface is defined in the header `metric.h` and explained in the following sections.

2.1 `metric_error()`

2.1.1 Synopsis

```
char *metric_error(void);
```

2.1.2 Description

This function returns a NULL terminated string describing the last error encountered by any call to the library.

2.1.3 Parameters

There are no parameters to this function.

2.1.4 Return Value

Returns a NULL terminated string describing the last error message encountered by any call to the library.

2.2 metric_deserialise()

2.2.1 Synopsis

```
metric_t *metric_deserialise(char *buffer);
```

2.2.2 Description

This function accepts a NULL terminated buffer containing a serialised metric in clear text and de-serialises it into the a metric structure which is returned. When no longer required the returned metric structure must be freed using the function `metric_delete()`.

2.2.3 Parameters

- `buffer`
  This parameter contains the NULL terminated clear text serialised metric.

2.2.4 Return Value

On success a pointer to the metric structure is returned, otherwise NULL is returned and an error message can be obtained by calling `metric_error()`.

2.3 metric_serialise()

2.3.1 Synopsis

```
int metric_serialise(metric_t *metric, int len, unsigned char *buffer);
```

2.3.2 Description

This function accepts a metric and serialises it into the output `buffer` as clear text of maximum size `len` (including the NULL terminator).

2.3.3 Parameters

- `len`
  This parameter contains the size of the supplied `buffer` parameter.

- metric
  This parameter contains the de-serialised metric.
- buffer
  The metric will be serialised into buffer as clear text and NULL terminated.

2.3.4 Return Value

On success the actual length of the buffer used is returned and the buffer will contain the serialised metric as a NULL terminated clear text string, otherwise -1 is returned and an error message can be obtained by calling metric_error(), in which case the output buffer content should not be trusted or assumed to be unchanged from before the function was called.

2.4 metric_last_serialised()

2.4.1 Synopsis

```
char *metric_last_serialised(void);
```

2.4.2 Description

This function returns a static buffer of the last metric serialised by the function metric_serialise().

2.4.3 Parameters

There are no parameters to this function.

2.4.4 Return Value

Returns a static buffer of the last metric serialised by the function metric_serialise().

2.5 metric_delete()

2.5.1 Synopsis

```
void metric_delete(metric_t *metric);
```

2.5.2 Description

This function allows an application to delete the metric structure and any related resources in one call.

2.5.3 Parameters

- `metric`
  This parameter contains the de-serialised metric.

2.5.4 Return Value

There is no return value from this function.

2.6 metric_open_dest_udp()

2.6.1 Synopsis

```
metric_dest_t *metric_open_dest_udp(char *host,int port);
```

2.6.2 Description

This function opens a UDP connection for sending metrics to a server and returns the metric destination structure. When no longer required the connection must be closed using the function `metric_close_dest()`.

2.6.3 Parameters

- `host`
  This parameter contains the server host *IPaddress*, which is either. a host name or the Internet notation of dots and numbers.
- `port`
  This parameter contains the server host port number.

2.6.4 Return Value

On success a pointer to the metric destination structure is returned, otherwise NULL is returned and an error message can be obtained by calling `metric_error()`.

2.7 metric_close_dest()

2.7.1 Synopsis

```
void metric_close_dest(metric_dest_t *dest);
```

2.7.2 Description

This function closes the connection to the server, deletes the destination structure and frees all related resources.

2.7.3 Parameters

- `metric_dest`
  This parameter contains a pointer to the metric destination structure created by the function `metric_open_dest_udp()`.

2.7.4 Return Value

There is no return value from this function.

2.8 metric_create()

2.8.1 Synopsis

```
metric_t *metric_create(char *name,char *title,char *source,char *group,
      metric_dest_t *metric_dest,int send_interval_secs);
```

2.8.2 Description

This function creates a metric structure of type `METRIC_IS_VALUES` from the supplied parameters.

2.8.3 Parameters

- `name`
  This parameter contains the name of the metric.
- `title`
  This parameter contains the title of the metric.

- source
  This parameter contains the source (originator) of the metric.
- metric_dest
  This parameter contains a pointer to the metric destination structure. created by the function metric_open_dest_udp().
- send_interval_secs
  This parameter contains the interval in seconds between sending subsequent metrics to the server. In other words this defines the frequency that metrics are sent to the server.

2.8.4 Return Value

On success a pointer to the metric structure is returned, otherwise NULL is returned and an error message can be obtained by calling metric_error().

2.9 metric_check_for_send()

2.9.1 Synopsis

```
int metric_check_for_send(metric_t *metric);
```

2.9.2 Description

This function checks if a metric needs to be sent.

The metric will be sent to the server[1] if the interval seconds[2] have elapsed since the last metric was sent.

2.9.3 Parameters

- metric
  This parameter contains a pointer to the metric structure.

2.9.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling metric_error().

[1] as specified in parameter metric_dest on the call to metric_create()
[2] as specified in parameter send_interval_secs on the call to metric_create()

2.10 metric_send()

2.10.1 Synopsis

```
int metric_send(metric_t *metric);
```

2.10.2 Description

The metric will be sent to the server as specified in parameter `metric_dest` on the call to `metric_create()`.

2.10.3 Parameters

- `metric`
  This parameter contains a pointer to the metric structure.

2.10.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

2.11 metric_update_count_relative()

2.11.1 Synopsis

```
int metric_update_count_relative(metric_t *metric,int count);
```

2.11.2 Description

This function increments the `count` value of the supplied metric by the value of the parameter `count`.

The metric will be sent to the server[3] if the interval seconds[4] have elapsed since the last metric was sent.

[3] as specified in parameter `metric_dest` on the call to `metric_create()`
[4] as specified in parameter `send_interval_secs` on the call to `metric_create()`

2.11.3 Parameters

- `metric`
  This parameter contains a pointer to the metric structure.
- `count`
  This parameter contains the count increment value.

2.11.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

2.12 metric_update_count_absolute()

2.12.1 Synopsis

```
int metric_update_count_absolute(metric_t *metric,int count);
```

2.12.2 Description

This function sets the `count` value of the supplied metric to the value of parameter `count`.

The metric will be sent to the server[5] if the interval seconds[6] have elapsed since the last metric was sent.

2.12.3 Parameters

- `metric`
  This parameter contains a pointer to the metric structure.
- `count`
  This parameter contains a count to which the `count` value of the metric is to be set.

[5] as specified in parameter `metric_dest` on the call to `metric_create()`
[6] as specified in parameter `send_interval_secs` on the call to `metric_create()`

2.12.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

2.13 metric_update_value()

2.13.1 Synopsis

```
int metric_update_value(metric_t *metric,long double value);
```

2.13.2 Description

This function increments the `count` value of the supplied metric by 1, adds the value of the parameter `value` to the `sum_values` value of the metric and adds the square of the value of the parameter `value` to the `sum_sq_values` value of the metric.

The metric will be sent to the server[7] if the interval seconds[8] have elapsed since the last metric was sent.

2.13.3 Parameters

- `metric`
  This parameter contains a pointer to the metric structure.
- `value`
  This parameter contains a value to be added to the `sum_values` value of the metric and the square of this value to be added to the `sum_sq_values` value of the metric.

2.13.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

[7] as specified in parameter `metric_dest` on the call to `metric_create()`
[8] as specified in parameter `send_interval_secs` on the call to `metric_create()`

2.14 metric_set_all_values()

2.14.1 Synopsis

```
int metric_set_all_values(metric_t *metric,int count,long double sum_values,
      long double sum_sq_values);
```

2.14.2 Description

This function sets the three values of the supplied metric to the parameters supplied.

The metric will be sent to the server[9] if the interval seconds[10] have elapsed since the last metric was sent.

2.14.3 Parameters

- `metric`
  This parameter contains a pointer to the metric structure.
- `count`
  The `count` value of the metric is set to this parameter.
- `sum_values`
  The `sum_values` value of the metric is set to this parameter.
- `sum_sq_values`
  The `sum_sq_values` value of the metric is set to this parameter.

2.14.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

2.15 metric_send_event()

2.15.1 Synopsis

```
int metric_send_event(char *name,char *title,char *source,char *group,
      metric_dest_t *metric_dest,char *event);
```

[9] as specified in parameter `metric_dest` on the call to `metric_create()`
[10] as specified in parameter `send_interval_secs` on the call to `metric_create()`

2.15.2 Description

This function sends a metric of type METRIC_IS_EVENT generated from the supplied parameters.

2.15.3 Parameters

- name
  This parameter contains the name of the metric.
- title
  This parameter contains the title of the metric.
- source
  This parameter contains the source (originator) of the metric.
- metric_dest
  This parameter contains a pointer to the metric destination structure created by the function metric_open_dest_udp().
- event
  This parameter contains a string describing the event.

2.15.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling metric_error().

2.16 metric_send_event_class()

2.16.1 Synopsis

```
int metric_send_event_class(metric_event_class_t event_class,char *name,
      char *title,char *source,char *group,metric_dest_t *dest,char *event);
```

2.16.2 Description

This function sends a metric of type METRIC_IS_EVENT generated from the supplied parameters.

Note this function is the same as the previous event send function (refer to section 2.15 on page 13) with the addition of the event class parameter.

2.16.3 Parameters

- `event_class`
  This parameter contains the class of the event. This must be a valid class of event - refer to `metric_event_class_t` in appendix A on page 23 for valid class values.
- `name`
  This parameter contains the name of the metric.
- `title`
  This parameter contains the title of the metric.
- `source`
  This parameter contains the source (originator) of the metric.
- `metric_dest`
  This parameter contains a pointer to the metric destination structure created by the function `metric_open_dest_udp()`.
- `event`
  This parameter contains a string describing the event.

2.16.4 Return Value

On success zero is returned, otherwise -1 is returned and an error message can be obtained by calling `metric_error()`.

3 Serialised Metric Grammar

This section documents and explains the syntax of the clear text serialised metric. It is only included for completeness as a serialised metric should always be created through the library from the abstract type.

3.1 Elements

The elements of the metric grammar comprise reserved words, identifiers, string literals, numbers and integers. The serialised metric grammar is free format and white spaces have no grammatical meaning except where they might appear within string literals.

3.1.1 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of the serialised metric. The reserved words are:

```
class       count           elapsed    event
group       interval_count  instances  metric
originator  port            pvalue     pvalues
rate        response        send_host  send_interval
seq         time            title      type
value       values          valuesl
```

3.1.2 Identifiers

Identifiers are case sensitive and they start with a letter which can be followed by any number of letters, digits, decimal point '.' or the under-score character.

Examples:

```
ifInOctets.1  Sessions
```

3.1.3 Strings

Strings are:

- any sequence of characters (except double quotes and the newline character) enclosed by double quotes.
- any sequence of characters (except single quotes and the newline character) enclosed by single quotes.

Examples:

```
"GigabitEthernet0/0 In Octets"
'ATM was disconnected'
```

3.1.4 Integers

An integer consists of a nonempty sequence of decimal digits.

Examples:

```
1234
0
```

3.1.5 Numbers

A number consists of a nonempty sequence of decimal digits that

- possibly contains a radix character (decimal point '.').
- is optionally followed by a decimal exponent; consisting of an 'E' or 'e' followed by an optional plus or minus sign followed by a nonempty sequence of decimal digits that indicates multiplication by a power of 10.

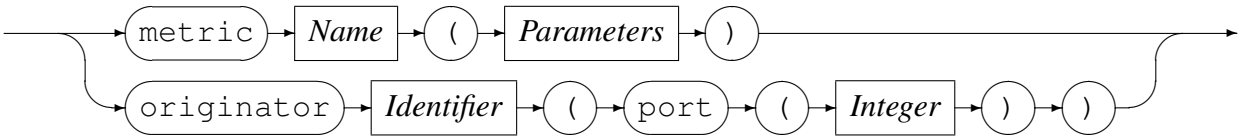
Examples:

```
1234
0.001
1.2
123.45E-12
```

3.2 Syntax

3.2.1 *Input*

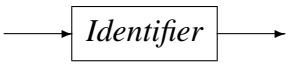
Input



A Metric input consists of a data metric or an originator definition. If it is an originator definition the metric type is set to `METRIC_IS_ORIGINATOR`. If it is a data metric then the *MetricType* determines the actual metric type (refer to section 3.2.5 on page 19).

3.2.2 *Name*

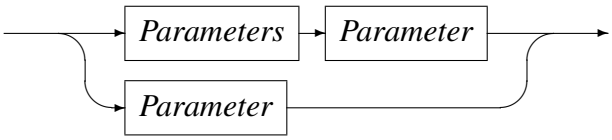
Name



Name is the name of the metric.

3.2.3 *Parameters*

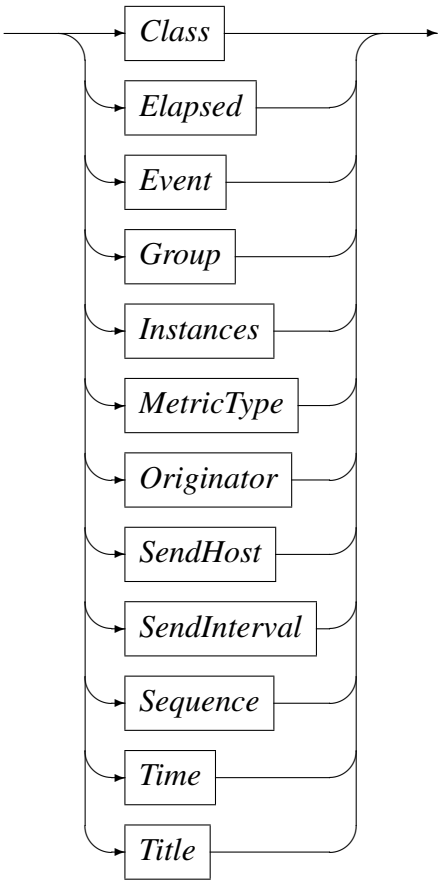
Parameters



A serialised metric is specified by a number of parameters in any order some of which are mandatory.

3.2.4 *Parameter*

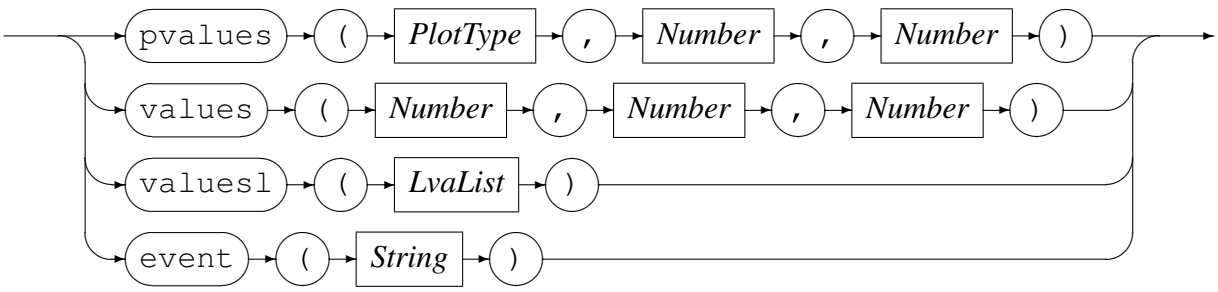
Parameter



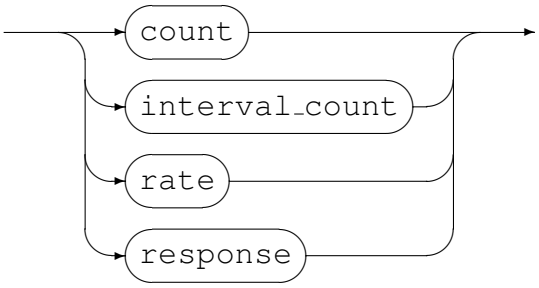
The metric parameters are explained in more detail in the following sections.

3.2.5 *MetricType*

MetricType



PlotType



This parameter holds the metric data and specifies the type of metric. The types of metric and the values it may hold are as follows:

pvalues: For a *pvalues* type metric the first parameter is the plot type, followed by the plot values for the X and Y axis.

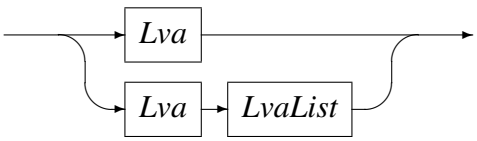
values: For a *values* type metric the three numbers represent count, sum of the values and sum of the squares.

lvalues: A labelled values list holds a parameter list of keywords and values and is explained below in section 3.2.6 on page 20.

event: An *event* metric holds only a string describing the event.

3.2.6 *LvaList*

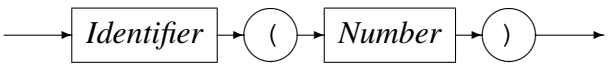
LvaList



A labelled values list metric has one or more values each with a unique name.

3.2.7 *Lva*

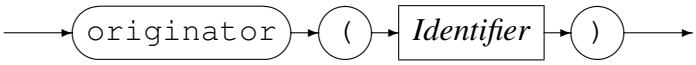
Lva



A labelled value consists of the unique name (for this metric only) and the value it is currently set to.

3.2.8 *Originator*

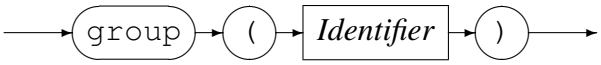
Originator



For a data metric this holds the name of the originator of the metric.

3.2.9 *Group*

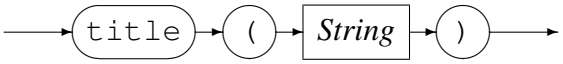
Group



This mandatory parameter names the group or category the metric belongs to.

3.2.10 *Title*

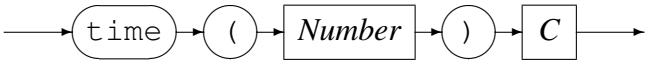
Title



This allows the metric to have a title that may describe it in terms familiar with the users of the metric.

3.2.11 *Time*

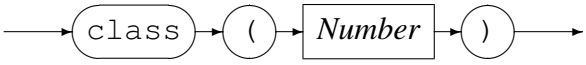
Time



Time is a mandatory parameter and is the time at which the metric was created, measured in seconds since the Epoch (00:00:00 UTC, January 1, 1970).

3.2.12 *Class*

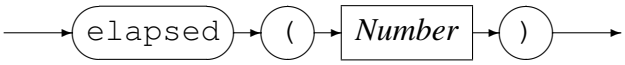
Class



This is the class of event and is only for a metric of type event.

3.2.13 *Elapsed*

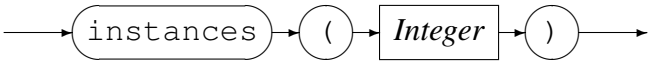
Elapsed



This is the time in seconds between producing this metric and the very first one.

3.2.14 *Instances*

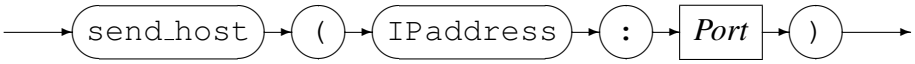
Instances



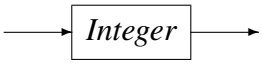
This specifies the number of instances of the metric producer if available.

3.2.15 *SendHost*

SendHost



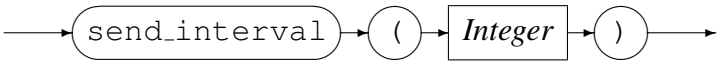
Port



This specifies the address of the server that the metric is send to.. *IPaddress* can be specified as a host name or by using the Internet notation of dots and numbers.

3.2.16 *SendInterval*

SendInterval



This specifies the interval in seconds that the metric is sent to the server.

A Header file `metric.h`

```
#ifndef METRIC_H
#define METRIC_H

  /* File: metric.h
   *
   * This header file describes the interface to the Code Magus Limited metric
   * library. The library implements an abstract data type describing a metric.
   * A single metric may be serialised into a buffer for consistent delivery to
   * other applications (often across a network) or de-serialised from a buffer
   * back into the data structure for consumption by an application.
   *
   * Copyright (c) 2010 Code Magus Limited. All rights reserved.
   *
   */

  /*
   * $Author: hayward $
   * $Date: 2020/08/25 10:27:22 $
   * $Id: metric.h,v 1.24 2020/08/25 10:27:22 hayward Exp $
   * $Revision: 1.24 $
   * $State: Exp $
   *
   * $Log: metric.h,v $
   * Revision 1.24  2020/08/25 10:27:22  hayward
   * Add source IP address to the metric struct.
   * It needs to be parsed when de-serialising
   * and written when serialising. It also needs
   * to be freed when a metric is deleted.
   *
   * Revision 1.23  2017/04/18 17:47:36  hayward
   * Add functionality to allow a user
   * to not have the timestamp in the
   * metric about to be sent updated
   * automatically. It is then the users
   * responsibility to update this time
   * so that the metric is sent in the
   * correct interval timing. This allows
   * post processors to either replay
   * metrics or use an original time from
   * a log as the timestamp for a newly
   * generated metric.
   *
   * Revision 1.22  2016/02/03 15:44:08  janvlok
   * Use new GMP usage directives.
   *
   * Revision 1.21  2016/02/03 14:32:07  janvlok
   * Removed c++ comments
   *
   * Revision 1.20  2016/02/03 14:07:07  janvlok
   * Implemented metric categories
```

```
*
* Revision 1.19  2015/12/07 13:52:10  janvlok
* First phase of category and units
*
* Revision 1.18  2015/01/09 10:31:12  janvlok
* Implement NA for metric values
*
* Revision 1.17  2014/07/08 15:51:05  hayward
* Make the serial buffer max size a
* C hash define to ease changing and
* using it in programs calling metric.
*
* Revision 1.16  2013/11/04 16:22:22  hayward
* Add TCP transport for sending metric events.
*
* Revision 1.15  2011/11/28 07:30:04  janvlok
* Remove qoutes from event string
*
* Revision 1.14  2011/11/02 09:46:17  hayward
* Delete debug comments.
*
* Revision 1.13  2011/10/07 09:50:37  janvlok
* Added event class and metric sequence number
*
* Revision 1.12  2011/08/05 07:05:49  janvlok
* Added function metric_check_for_send()
*
* Revision 1.11  2011/05/06 08:48:15  janvlok
* New Functions
*
* Revision 1.10  2011/04/27 11:35:11  janvlok
* added _values to sum and sum_sq
*
* Revision 1.9  2011/04/27 07:16:17  janvlok
* New functions implemented
*
* Revision 1.8  2011/01/11 10:45:02  hayward
* sys/time.h is needed on MVS but does not exist
* on Windows and is allowed on Linux/Unix.
*
* Revision 1.7  2010/11/09 12:00:36  janvlok
* Qualifieed for windows
*
* Revision 1.6  2010/09/13 07:13:41  janvlok
* Implemented plot metric
*
* Revision 1.5  2010/09/06 15:37:35  hayward
* Add sys/time.h for "struct timeval"
* for the picky MVS compiler.
*
* Revision 1.4  2010/08/31 10:40:08  janvlok
* Made type_name obsolete
```

```
   *
   * Revision 1.3  2010/08/27 06:50:04  janvlok
   * Fixed line overflow in the comments
   *
   * Revision 1.2  2010/08/25 13:55:29  hayward
   * Corrct and improve the specification of the library.
   *
   * Revision 1.1.1.1  2010/08/25 08:55:36  hayward
   * Add metric header to CVS.
   *
   */

static char cvs_metric_h[] =
   "$Id: metric.h,v 1.24 2020/08/25 10:27:22 hayward Exp $";

#include <openssl/bn.h>
#include <time.h>
#ifndef WIN32
/* Required for MVS */
#include <sys/time.h>
#endif

#ifdef METRIC_GMP
#include <gmp.h>
#endif

  /*
   * Defines and constants:
   */

#define METRIC_DEFAULT_MPF_PREC 256 /* GMP float mantissa precision in bits */
typedef enum
   {
   MCATEGORY_NONE = 0,           /* NA */
   MCATEGORY_ABSOLUTE,           /* metric_update_count_absolute() */
   MCATEGORY_COUNTER,            /* metric_update_counter() */
   MCATEGORY_RELATIVE,           /* metric_update_count_relative() */
   MCATEGORY_SAMPLE,             /* metric_update_value() */
   MCATEGORY_VALUES,             /* metric_set_all_values() */
   } metric_category_t;

typedef enum
   {
   METRIC_IS_NOTYPE = 0,         /* Type has not been set */
   METRIC_IS_VALUES,             /* Data are count, sum and sum of squares */
   METRIC_IS_LVALUES,            /* Data are an unbounded set of labelled values
                                    */
   METRIC_IS_EVENT,              /* Datum is an event (single string) */
   METRIC_IS_ORIGINATOR,         /* Datum is originator's contact information */
   METRIC_IS_PLOT_VALUES         /* Data are plot type, X axis and Y axis */
   } metric_type_t;
```

```
typedef enum
      {
      METRIC_PLOT_NOTYPE = 0,     /* Type has not been set */
      METRIC_PLOT_COUNT,          /* plot count */
      METRIC_PLOT_INTERVAL_COUNT,/* plot count in last interval */
      METRIC_PLOT_RATE,           /* plot rate */
      METRIC_PLOT_RESPONSE        /* plot response */
      } metric_plot_type_t;

typedef enum
      {
      METRIC_EVENT_IS_LOW = 0,    /* low  */
      METRIC_EVENT_IS_MEDIUM,     /* medium  */
      METRIC_EVENT_IS_HIGH        /* high  */
      } metric_event_class_t;

#define METRIC_SERIAL_BUF_MAX_SIZE 32000
  /*
   * Types and Structures:
   */

typedef struct metric metric_t;
typedef struct metric_value metric_value_t;
typedef struct metric_lvalue metric_lvalue_t;
typedef struct metric_lva metric_lva_t;
typedef struct metric_plot_va metric_plot_va_t;
typedef struct metric_dest metric_dest_t;

  /* Structure metric_t describes a metric. It may only have one type which
   * determines the values in the data member.
   */

struct metric
   {
   metric_type_t type;            /* metric type */
   char *name;                    /* name */
   char *title;                   /* title */
   char *source;                  /* name of creator of metric */
   char *group;                   /* group name */
   struct timeval time;           /* time metric was created */
   long double elapsed;           /* elapsed time in seconds set by originator */
   int instances;                 /* number of instances */
   char *send_host_ipaddr;        /* host IP address for delivery */
   int send_host_port;            /* host port for delivery */
   int send_interval;             /* send interval in seconds */
   union
      {
      metric_value_t *value;      /* values for METRIC_IS_VALUES */
      metric_lvalue_t *lvalue;    /* lvalues for METRIC_IS_LVALUES */
      metric_plot_va_t *pvalue;   /* plot values for METRIC_IS_PLOT_VALUES */
      char *event;                /* message for METRIC_IS_EVENT */
      int ori_cmd_port;           /* port for METRIC_IS_ORIGINATOR */
```

```
      } data;                    /* data parsed */

   metric_dest_t *dest;          /* server destination address */
   struct timeval curr_tv;       /* current time */
   long double curr_time;        /* current time */
   long double send_time;        /* time to send the metric time */
   long double create_time;      /* time metric was created */
   long double updated_time;     /* time metric was last updated */
   int sequence;                 /* metric sequence number */
   metric_event_class_t event_class;/* event class */
   metric_category_t category;   /* metric category */
   char *units;                  /* metric units */
   char *source_ipaddr;          /* IP address of host sending the metric */
   };

  /* Structure metric_value_t describes the values for metric type
   * METRIC_IS_VALUES and includes the common statistical values of count, sum
   * (of time since previous metric) and the sum of the squares.
   */

struct metric_value
   {
   long double count;            /* count - first value */
   long double sum_values;       /* sum of values */
   long double sum_sq_values;    /* sum of the squares */
   char count_is_na;             /* count - Not Applicable */
   char sum_values_is_na;        /* sum_values - Not Applicable */
   char sum_sq_values_is_na;     /* sum_sq_values - Not Applicable */
   char use_mpf;                 /* set if GMP float counters in use */
#ifdef METRIC_GMP
   mpf_t mpf_count;              /* GMP float: count - first value */
   mpf_t mpf_sum_values;         /* GMP float: sum of values */
   mpf_t mpf_sum_sq_values;      /* GMP float: sum of the squares */
#endif
   };

  /* Structure metric_lvalue_t describes the values for metric type
   * METRIC_IS_LVALUES. The labelled values are held in a singly-linked list and
   * nlvalues specifies how many entries are present.
   */

struct metric_lvalue
   {
   metric_lva_t *head;           /* label values list */
   int nlvalues;                 /* number of label values */
   };

struct metric_lva
   {
   metric_lva_t *next;           /* next in the list */
   char *label;                  /* label for the value */
   long double value;            /* value */
```

```
   };

  /* Structure metric_plot_va_t describes the values for metric type
   * METRIC_IS_PLOT_VALUES and includes the plot type, X axis value
   * Y axis value..
   */

struct metric_plot_va
   {
   metric_plot_type_t plot_type; /* plot type */
   long double xva;              /* X axis value */
   long double yva;              /* Y axis value */
   };

  /*
   * Exposed functions:
   */

  /* Function metric_error() will return a NULL terminated string describing the
   * last error message encountered by any call to the library.
   */

char *metric_error(void);

  /* Function metric_deserialise() accepts a NULL terminated buffer containing
   * a serialised metric and de-serialises it into the a metric structure which
   * is returned. When no longer required the returned metric structure must be
   * freed using the function metric_delete().
   * On success a pointer to the metric structure is returned, otherwise NULL is
   * returned and an error message can be obtained by calling metric_error().
   */

metric_t *metric_deserialise(char *buffer);

  /* Function metric_serialise() accepts a metric and serialises it into the
   * output buffer of maximum size len-1.
   * On success the actual length of the buffer used is returned and the buffer
   * will contain the serialised metric as a NULL terminated string, otherwise
   * -1 is returned and an error message can be obtained by calling
   * metric_error(), in which case the output buffer content should not be
   * trusted or assumed to be unchanged from before the function was called.
   */

int metric_serialise(metric_t *metric, int len, unsigned char *buffer);

  /* Function metric_last_serialised() returns a static buffer of the last
   * metric serialised by metric_serialise().
   */

char *metric_last_serialised(void);

  /* Function metric_delete() allows an application to delete the metric
```

```
   * structure and any related resources are freed.
   * There is no return value from this function.
   */

void metric_delete(metric_t *metric);

  /* Function metric_open_dest_udp() opens a UDP connection for sending of
   * metrics to a server.
   * *host id the server host IPaddress, which is either. a host name or
   * the Internet notation of dots and numbers, and port is the server
   * port number.
   * On success the created metric destination structure is returned, otherwise
   * NULL is returned and an error message can be obtained by calling
   * metric_error().
   */

metric_dest_t *metric_open_dest_udp(char *host,int port);

  /* Function metric_open_dest_tcp() opens a TCP connection for sending of
   * metrics to a server.
   * *host id the server host IPaddress, which is either. a host name or
   * the Internet notation of dots and numbers, and port is the server
   * port number.
   * On success the a connection to the remote host is made and the created
   * metric destination structure is returned, otherwise NULL is returned and an
   * error message can be obtained by calling metric_error().
   */

metric_dest_t *metric_open_dest_tcp(char *host,int port);

  /* Function metric_close_dest() closes the connection to the server, delete
   * the destination structure and free all related resources.
   * There is no return value from this function.
   */

void metric_close_dest(metric_dest_t *dest);

  /* Function metric_create() creates a metric of type METRIC_IS_VALUES
   * from the parameters supplied:
   * *name is the the name of the metric, *title the title, *source is
   * the originator of the metric, *group is the group it belongs to.
   * *metric_dest is the server to send the metric to
   * and send_interval_secs is the frequency in seconds for sending.
   * to the server.
   * On success the created metric structure is returned, otherwise
   * NULL is returned and an error message can be obtained by calling
   * metric_error().
   */

metric_t *metric_create(char *name,char *title,char *source,char *group,
      metric_dest_t *metric_dest,int send_interval_secs);
```

```
  /* Function metric_check_for_send() checks if the metric is due for sending,
   * if so, send it to the server. The frequency and server address are
   * supplied as parameters on creating the metric, using the function
   * metric_create(),
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_check_for_send(metric_t *metric);

  /* Function metric_send() send the metric to the server,
   * The server address are supplied as parameters on creating the metric,
   * using the function metric_create(),
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_send(metric_t *metric);

  /* Function metric_sendto() send a serialised metric to the server.
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_sendto(metric_dest_t *dest,char *buf,int len);

  /* Function metric_send_tcp() send a serialised metric to the server using
   * TCP/IP.
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_send_tcp(metric_t *metric);

  /* Function metric_update_count_relative() increments the count value of
   * the metric from the supplied parameter count.
   * The metric category MCATEGORY_RELATIVE
   *
   * The metric will be sent to server when it is due for sending. The frequency
   * and server address are supplied as parameters on creating the metric, using
   * the function metric_create(),
   * Note if the metric is to be sent, it will be done before it is updated.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_update_count_relative(metric_t *metric,int count);

  /* Function metric_update_count_absolute() sets the count value of
   * the metric to the value of the supplied parameter count.
   * The metric category MCATEGORY_ABSOLUTE
```

```
   *
   * The metric will be sent to server when it is due for sending. The frequency
   * and server address are supplied as parameters on creating the metric, using
   * the function metric_create(),
   * Note if the metric is to be sent, it will be done before it is updated.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_update_count_absolute(metric_t *metric,int count);

  /* Function metric_update_value() increments the count value of the metric
   * by 1, parameter value is added to sum value of the metric and the square of
   * the value is added to sum_sq value of the metric.
   * The metric category is MCATEGORY_SAMPLE.
   *
   * The metric will be sent to server when it is due for sending. The frequency
   * and server address are supplied as parameters on creating the metric, using
   * the function metric_create(),
   * Note if the metric is to be sent, it will be done before it is updated.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_update_value(metric_t *metric,long double value);

  /* Function metric_update_counter() increments the count value of the metric
   * by 1. The delta time (milliseconds since previus update) is added to the
   * sum value of the metric and the square of the delta is added to sum_sq
   * value of the metric.
   * The metric category is MCATEGORY_COUNTER.
   *
   * The metric will be sent to server when it is due for sending. The frequency
   * and server address are supplied as parameters on creating the metric, using
   * the function metric_create(),
   * Note if the metric is to be sent, it will be done before it is updated.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_update_counter(metric_t *metric);

  /* Function metric_set_all_values() set the three values of the metric to
   * to the supplied parameters count, sum and sum_sq respectively.
   * The metric category  MCATEGORY_VALUES
   *
   * The metric will be sent to server when it is due for sending. The frequency
   * and server address are supplied as parameters on creating the metric, using
   * the function metric_create(),
```

```
   * Note if the metric is to be sent, it will be done before it is updated.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_set_all_values(metric_t *metric,int count,long double sum,
      long double sum_sq);

  /* Function metric_send_event() send a metric of type METRIC_IS_EVENT
   * as per the parameters supplied:
   * *name is the the name of the metric, *title the title, *source is
   * the originator of the metric, *group is the group it belongs to,
   * *metric_dest is  the destination for sending the metric and *event
   * is a string describing the event.
   *
   * On success zero is returned, otherwise -1 is returned and an error message
   * can be obtained by calling metric_error().
   */

int metric_send_event(char *name,char *title,char *source,char *group,
      metric_dest_t *metric_dest,char *event);
int metric_send_event_class(metric_event_class_t event_class,char *name,
      char *title,char *source,char *group,metric_dest_t *dest,char *event);

  /* Function metric_set_flag() allows a caller to set various valid flags in
   * order to control logic processing within the library. See the flags below
   * for their values and descriptions. Any flag outside the library defined
   * valid flags is silently ignored.
   */

int set_flag(metric_t *metric, unsigned int flag);

  /* FLAG METRIC_DISABLE_AUTO_TS disables the automatic updating of the
   * timestamp and sending of the metric when a value is updated. This means
   * that it is up to the caller to send the metric when they want to and to
   * make sure the timestamps are all set.
   * This is useful where events are post processed (ie not in real time) and
   * metrics created, but the timestamp of the metric should reflect the
   * original event timing.
   */
#define METRIC_DISABLE_AUTO_TS   0X00000001

#endif /* METRIC_H */
```