**MySQL  Connector/Arduino**

Dr.  Charles  Bell  January  2016  v1.1.1a

Have  you  ever  wanted  to  connect  your  Arduino  project  to  a  database  to  either  store  the  data  you’ve  collected  or  retrieve  data  saved  to  trigger  events  in  your  sketch?  Well,  now  you  can  connect  your  Arduino  project  directly  to  a  MySQL  server  without  using  an  intermediate  computer  or  a  web-­‐  or  cloud-­‐based  service.  Having  direct  access  to  a  database  server  means  you  can  store  data  acquired  from  your  project  as  well  as  check  values  stored  in  tables  on  the  server.

This  also  means  you  can  setup  your  own,  local  MySQL  server  to  store  your  data  further  removing  the  need  for  Internet  connectivity.  If  that  is  not  an  issue,  you  can  still  connect  to  and  store  data  on  a  MySQL  server  via  your  network,  Internet,  or  even  in  the  cloud!

The  MySQL  Connector/Arduino  is  a  library  that  permits  you  to  do  exactly  that  and  more!  This  document  is  intended  to  help  you  learn  about  the  MySQL  Connector/Arduino  library  and  discover  its  powerful  capabilities.  You  will  also  discover  important  insight  into  how  to  use  the  library  as  well  as  vital  troubleshooting  tips  to  overcome  problems.  Read  on  as  we  discover  the  capabilities  and  see  examples  of  how  this  library  works.

**Note**     *This  version  of  the  connector  is  a  major  revision  from  the  1.0  branch.  If  you  have  been  using  the  1.0.X  version,  you  may  want  to  read  the  section  entitled,  “Changes  from  Previous  Versions”  to  see  what  has  changed.*

**Getting Started**

If  you  have  used  some  of  the  other  methods  of  storing  data  from  an  Arduino  such  as  writing  data  to  flash  memory  (e.g.  a  secure  digital  card)  or  an  EEPROM  device,  you  probably  had  to  write  additional  code  to  read  that  data  for  later  use.  If  you  wanted  to  use  that  data  on  a  device  other  than  the  Arduino,  you  probably  had  to  manually  copy  the  data  in  order  to  use  it.  Using  a  database  to  store  the  data  can  eliminate  the  manual  data  copy  and  extraction  method  altogether.  Similarly,  if  your  project  is  such  that  you  cannot  or  do  not  want  to  connect  to  the  Internet  to  save  your  data,  the  ability  to  write  to  a  local  database  server  solves  that  problem  as  well.

Saving  your  data  in  a  database  will  not  only  preserve  the  data  for  analysis  at  a  later  time,  it  also  means  your  project  can  feed  data  to  more  complex  applications  that  make  use  of  the  data.  Better  still,  if  you  have  projects  that  use  large  data  values  for  calculations  or  lookups,  you  can  store  the  data  on  the  server  and  retrieve  only  the  data  you  need  for  the  calculation  or  operation  all  without  taking  up  large  blocks  of  memory  on  your  Arduino.  Clearly,  this  opens  a  whole  new  avenue  of  Arduino  projects!

The  technology  is  named  Connector/Arduino  (for  use  in  this  document  simply,  the  connector).  The  connector  manages  the  MySQL  client  communication  protocol  in  a  library  built  for  the  Arduino  platform.  In  fact,  all  of  the  mechanisms  for  communicating  with  the  MySQL  server  are  hidden  so  you  do  not  need  to  learn  the  minutia  of  the  protocol.

**Note**       *Henceforth  we  refer  to  Connector/Arduino  when  discussing  general  concepts  and  features  and  refer  to  the  actual  source  code  using  the  term  the  Connector/Arduino  library  or  simply  the  library.*

Sketches  (programs)  written  to  use  the  library  permit  you  to  encode  SQL  statements  to  insert  data

and  run  small  queries  to  return  data  from  the  database  (e.g.  using  a  lookup  table).

You  may  be  wondering  how  a  memory  and  processing  limited  microcontroller  can  possibly  support  code  to  insert  data  into  a  MySQL  server.  We  can  do  this  because  the  protocol  for  communicating  with  a  MySQL  server  is  not  only  well  known  and  documented,  but  also  intentionally  designed  to  be  lightweight.  Which  is  one  of  the  small  details  that  make  MySQL  attractive  for  embedded  developers.

In  order  to  communicate  with  MySQL,  the  Arduino  must  be  connected  to  the  MySQL  server  via  a  network.  To  do  so,  the  Arduino  must  use  an  Ethernet  or  WiFi  shield  and  be  connected  to  the  same  Ethernet  network  as  the  database  server.  In  fact,  the  library  is  compatible  with  most  new  Arduino  Ethernet  and  compatible  clone  shields  that  support  the  standard  Ethernet  library.

**Hardware Requirements**

The  connector  requires  an  Arduino  or  Arduino  clone  with  at  least  32k  of  memory.  If  you  are  using  an  older  Arduino  like  the  Duemilanove,  be  sure  you  have  the  version  that  uses  the  ATmega328p  processor.

If  your  sketch  is  more  than  a  few  lines  long,  you  are  using  a  lot  of  libraries,  have  a  lot  of  sensors  attached,  or  want  to  complex  queries,  you  should  consider  using  one  of  the  larger  (as  in  memory,  not  physical  size)  boards  such  as  the  Mega  or  Due.  We  will  see  why  this  is  so  in  a  later  section.

The  connector  also  requires  the  Arduino  Ethernet  shield  or  equivalent.  This  is  because  the  library  references  the  Ethernet  library  written  for  the  Ethernet  shield.  If  you  have  some  other  form  of  Ethernet  shield,  or  the  Ethernet  shield  you  are  using  uses  a  different  library,  you  will  have  to  make  a  slight  modification  to  the  library  to  use  it.

Finally,  the  connector  is  written  specifically  for  the  Arduino  Ethernet  and  WiFi  shields  or  modules  that  are  compatible  with  the  Ethernet  class  included  with  the  Arduino  IDE.  If  you  have  another  shield  or  module  that  requires  an  additional  library,  it  is  likely  it  will  not  work  with  this  connector.  Only  those  shields  that  use  the  Arduino-­‐supplied  Ethernet  class  will  work.

**Caution** *Compatibility  in  this  sense  means  you  can  use  any  shield  or  module  that  implements  the  same  class  signature  (methods)  as  the  Arduino-­‐supplied  Ethernet  class.  If  you  want  to  use  the  connector  with  another  library,  you  will  have  to  write  an  intermediate  class  to  translate  the  library  you  want  to  use  to  the  Ethernet  Client  class  signature.*

**A Note About Memory**

The  connector  is  implemented  as  an  Arduino  library.  While  the  protocol  is  lightweight,  the  library  does  consume  some  memory.  In  fact,  the  library  requires  about  20k  of  flash  memory  to  load.  Thus,  it  requires  the  ATmega328  or  similar  processor  with  32k  of  flash  memory.

That  may  seem  like  there  isn’t  a  lot  of  space  for  programming  your  sensor  node  but  as  it  turns  out  you  really  don’t  need  that  much  for  most  sensors.  If  you  do,  you  can  always  step  up  to  a  new  Arduino  with  more  memory.  For  example,  the  latest  Arduino,  the  Due,  has  512k  of  memory  for  program  code.  Based  on  that,  a  mere  20k  is  an  insignificant  amount  of  overhead.

However,  memory  limitations  can  easily  be  reached  when  you  use  additional  libraries.  Each  library  you  load  will  consume  memory  thereby  reducing  the  available  memory  for  dynamic  variables.  The  connector  must  allocate  memory  to  store  the  query  being  sent  (as  a  static  string)  as  well  as  the  results  returned  (dynamic  memory).  Thus,  if  you  have  several  queries  you  want  to  send,  each  one  of

those  will  require  space  and  if  you  return  rows  from  a  query,  each  row  requires  space.  A  combination

of  these  along  can  cause  a  moderately  complex  sketch  on  an  Uno  to  run  out  of  space.

You  can  do  a  lot  to  mitigate  this  problem.  You  can  use  a  board  with  more  memory,  reduce  the  number  of  variables,  reduce  the  size  of  the  rows  returned  (by  specifying  a  list  of  columns  instead  of  SELECT \*),  and  limit  the  use  of  libraries  and  unnecessary  code.  But  the  most  important  task  you  can  do  is  to  check  your  sketch  for  memory  leaks.

Memory  leaks  will  cause  your  sketch  to  lockup  when  it  runs  out  of  memory.  See  the  FAQ  section  below  for  more  details.  As  a  rule,  I  suggest  leaving  at  least  800  bytes  of  memory  available  for  dynamic  variables.  You  can  see  this  when  you  compile  your  sketch  as  shown  below.

Sketch uses 20,654 bytes (64%) of program storage space. Maximum is 32,256 bytes.

Global variables use 1,186 bytes (57%) of dynamic memory, leaving 862 bytes for local variables. Maximum is 2,048 bytes.

Here  we  see  the  sketch  leaves  only  862  bytes  for  local  variables.  This  should  be  sufficient  for  most

small  sketches  that  simply  write  data  to  the  database.

However,  consider  what  would  happen  if  you  wanted  to  retrieve  a  row  from  the  database  that  was

400  bytes  in  length.  The  connector  would  need  to  allocate  memory  for  that  row  and  therefore  leave  about  450  bytes  left.  Considering  the  Arduino  uses  this  memory  (e.g.  the  stack),  this  isn’t  enough  memory  to  permit  the  sketch  to  run  properly.  The  end  result  is  the  Arduino  will  likely  hang.

Consider  also  a  case  where  you  fail  to  release  the  dynamic  memory  allocated.  In  this  case,  consider  a  case  where  we  want  to  retrieve  a  very  small  amount  of  data  –  say  about  40  bytes.  The  sketch  will  likely  run  fine  for  some  time  depending  on  how  often  we  retrieve  the  data  but  each  time  it  does,  40  more  bytes  will  be  allocated  and  not  returned  thus  after  about  10  or  so  queries,  the  Arduino  will  run  out  of  memory  and  freeze.

**Networking Hardware**

Your  networking  hardware  should  be  the  usual  and  normal  devices  typically  found  in  a  home  or  small  office.  That  is,  you  should  have  some  sort  of  router  or  access  point  that  permits  you  to  connect  your  Ethernet  or  WiFi  shield  to  your  network.

For  example,  a  typical  wireless  access  port  or  cable  modem  will  have  additional  Ethernet  ports  that  you  can  use  to  connect  an  Ethernet  shield  using  a  normal  Ethernet  (CAT5  or  similar)  cable.  Do  not  use  a  crossover  cable  unless  you  know  what  one  is  and  how  to  use  it.

Similarly,  if  using  a  WiFi  shield,  your  WiFi  router  should  permit  connections  with  the  security  protocols  supported  (see  [https://www.arduino.cc/en/Main/ArduinoWiFiShield  for  more  details).](http://www.arduino.cc/en/Main/ArduinoWiFiShield)

Furthermore,  your  MySQL  server  and  Arduino  must  reside  on  network  segments  that  are  reachable.  Ideally,  they  should  be  on  the  same  subnet  but  that  isn’t  a  hard  requirement.

If  you  are  not  certain  of  your  network  configuration  or  you  are  attempting  to  build  a  solution  in  a  laboratory  at  a  university,  college,  or  at  work,  you  should  seek  out  the  local  IT  support  to  help  you  configure  your  hardware  on  the  network.

**MySQL Server**

The  requirements  for  the  MySQL  server  for  use  with  the  Arduino  are  simple.  First  and  foremost,  you  must  ensure  you  setup  the  MySQL  server  to  permit  network  connections.  See  the  online  MySQL  reference  manual  for  more  details  about  platform-­‐specific  installation  and  setup  [(http://dev.mysql.com/doc](http://dev.mysql.com/doc/)/).

**Note**       The  connector  is  designed  to  work  with  MySQL  5.0  and  later  using  the  latest  client  protocols.  If  you  want  to  use  the  connector  with  a  newer,  more  secure  version  of  the  MySQL  server,  you  must  ensure  you  are  using  the  5.X  era  authentication  protocols.  The  connector  will  not  work  with  a  custom  authentication  protocol.

More  specifically,  you  must  ensure  your  MySQL  server  is  not  setup  to  bind  on  a  network  address  (by  commenting  out  bind\_address  in  my.cfg)  and  that  there  are  no  firewalls  or  port  blocking  software  to  prohibit  access  to  the  server.  For  example,  it  is  not  uncommon  for  aggressive  anti-­‐virus  and  firewall  software  to  block  access  to  port  3306  (the  default  listening  port  for  MySQL).

Finally,  the  connector  is  designed  to  work  with  the  MySQL  server.  It  does  not  work  with  other

database  servers.  Thus,  you  cannot  use  it  with  other  database  systems.

**User Accounts**

You  will  need  a  user  account  and  password  to  use  in  your  sketch.  While  we  are  not  necessarily  concerned  about  strict  security  protocols  (but  there  is  nothing  wrong  with  that)  as  the  user  and  password  will  be  hard  coded  in  the  sketch  (at  least,  in  the  examples  below  –  you  can  use  your  own,  more  secure  methods  if  you  prefer).

This  is  perhaps  the  first  mistake  users  make.  They  either  use  the  root  account  with  no  password  (not  advisable)  or  they  create  a  user  that  is  not  permitted  to  connect  to  the  database.  More  precisely,  MySQL  uses  a  combination  of  user  and  host  to  form  a  login.  Examine  the  following  statements.  Are  these  the  same  user  or  different  users?

CREATE USER [bob@localhost IDENTIFIED BY 'secret'; CREATE USER bob@'192.168.0.5' IDENTIFIED BY 'secret';](mailto:bob@localhost)

The  answer  may  surprise  you.  They  are  two  different  users  even  though  the  user  name  and  password  are  the  same!  One  is  allowed  to  connect  only  through  the  local  host  machine.  That  is,  the  user  must  be  connected  to  the  same  host  as  the  server.  The  other  is  allowed  to  connect  to  the  server  if  and  only  if  that  user  is  located  on  a  machine  with  an  IP  address  of  192.168.0.5.

Now  consider  your  Arduino  will  be  connected  via  an  Ethernet  (or  WiFi  shield)  –  see  below.  This  means  your  Arduino  will  receive  its  own  IP  address  and  any  user  connecting  to  the  server  must  be  validated  via  the  user  and  host  name  (IP).  For  example,  if  your  Arduino  is  assigned  the  address  of

192.168.0.11,  you  cannot  connect  using  either  of  the  user  accounts  created  above!  Furthermore,  if  your  Arduino  uses  DHCP  to  get  an  IP  address,  you  may  not  know  what  IP  address  is  given.  So  how  do  you  overcome  this?  Use  masking.

The  simplest  way  to  create  the  user  is  by  using  a  wildcard  for  the  host  name  as  follows.

CREATE USER bob@’%’ IDENTIFIED BY 'secret';

Of  course,  this  is  not  very  secure  since  the  user  can  connect  from  any  host,  but  it  will  get  you  where  you  need  to  be  and  is  sufficient  for  most  Arduino  projects.  However,  if  you  want  a  more  secure  user  account,  you  could  limit  the  hosts  to  a  subnet  as  follows.

CREATE USER bob@’192.168.0.%’ IDENTIFIED BY 'secret';

**Privileges**

You  will  also  need  to  give  the  user  access  to  whatever  database(s)  that  you  want  to  access.  I  won’t  go  into  all  of  the  details  here,  but  suffice  to  say  you  need  to  grant  permissions  to  each  user  based  on  what  you  want  to  do.  For  example,  if  you  only  want  to  read  data,  a  simple  SELECT  permission  is  all  that  is  needed.  In  the  following  example,  I  go  to  the  other  extreme  and  grant  all  permissions  to  the  user.  This  is  Ok  since  I  am  both  restricting  the  user  to  a  specific  host  and  limiting  access  to  a  single  database.  Observe.

GRANT ALL ON test\_arduino.\* TO bob@’192.168.0.11’;

Here  I  have  give  the  user  access  and  all  permissions  to  any  object  in  the  test\_arduino  database.  Note  that  by  default  in  the  absence  of  any  other  GRANT  statement,  the  user  does  not  have  access  to  other  databases  on  the  system.  This  is  why  using  the  root  user  is  a  bad  idea.  If  you  make  a  mistake  in  your  sketch  and  update  or  delete  the  wrong  row  or  worse  delete  the  database,  you  will  permanently  loose  your  data.  Always  use  a  newly  create  user  account  with  minimal  permissions  for  your  Arduino  projects.

**Test Access**

Once  your  user  account  is  setup  and  you  have  granted  the  correct  permissions,  you  should  check  the  connection.  This  is  perhaps  the  one  thing  that  most  new  users  skip  and  assume  everything  will  work.  Furthermore,  they  make  the  mistake  of  testing  the  user  account  from  the  same  host  as  the  server.  This  will  mask  a  number  of  potential  pitfalls  and  is  not  the  best  test.

The  best  way  to  check  if  your  user  account  and  permissions  is  to  use  another  computer  to  login  to  the  MySQL  server.  Simply  connect  another  computer  to  the  same  network  and  open  the  mysql  client

and  attempt  to  connect.  If  you  do  not  have  another  computer  to  use,  you  can  force  the  mysql  client  to  use  the  network  to  connect  (as  opposed  to  socket  connections  by  default)  by  specifying  the  host  and  port  as  shown  below.  The  host  in  this  case  is  the  hostname  or  IP  of  the  MySQL  **server**,  **not**  the  Arduino.

mysql -ubob -psecret –h192.168.0.2 --port=3306

Once  you  have  connected,  try  accessing  the  database  and  attempt  any  operations  you  want  to  include  in  your  sketch.  Now  would  be  a  good  time  to  test  the  SQL  statements  you  plan  to  include  in  your  sketch.  Simply  type  them  in  and  run  them.  For  example,  try  inserting  some  dummy  data,  creating  objects,  selecting  rows  –  whatever  you  plan  to  do  in  your  sketch.  Not  only  will  this  verify  your  user  account  has  the  correct  permissions;  it  will  also  verify  your  SQL  statements  are  properly  written  and  thus  avoid  strange  errors  when  you  run  your  script.

Once  you  can  successfully  connect  and  have  verified  you  can  access  the  correct  database  objects,

reset  the  data  and  transfer  the  login  information  to  your  sketch  or  write  it  down  for  later  reference.

**How To Get MySQL Connector/Arduino**

The  easiest  way  to  start  using  the  connector  is  to  use  the  Library  Manager  to  download  and  install

the  connector.  Simply  open  the  Library  Manager  in  the  Arduino  IDE  from  the  *Sketch-­‐>Include  Library-­‐*

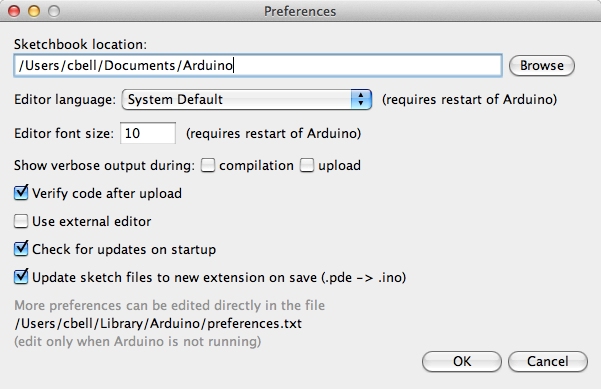
*>Manage  Libraries*  menu.  This  opens  the  Library  Manager.  In  the  filter  your  search  box,  enter  "MySQL"  then  choose  the  connector  and  click  Install.  In  seconds,  the  new  library  is  installed  and  ready  for  use.  You  can  also  revisit  the  Library  Manager  and  update  the  connector  library  whenever  a  new  version  is  released.  Thus,  you  can  more  easily  keep  your  libraries  up-­‐to-­‐date!

If  you  do  not  want  to  use  the  Library  Manager  (or  cannot  because  you’re  using  a  different  IDE  or  editor),  you  can  also  download  it  from  GitHub,  unzip  it,  and  place  it  in  your  Arduino/Libraries  folder.  You  can  download  Connector/Arduino  from  GitHub  (https://github.com/ChuckBell/MySQL\_Connector\_Arduino).  The  library  is  open  source,  licensed  as  GPLv2,  and  owned  by  Oracle  Corporation.  Thus,  any  modifications  to  the  library  that  you  intend  to  share  must  meet  the  GPLv2  license.

Once  you  have  downloaded  the  library,  you  need  to  copy  or  move  it  to  your  Arduino/Libraries  folder.  Place  the  folder  from  the  .zip  file  named  MySQL\_Connector\_Arduino  to  your  sketches  library  folder.

You  can  find  where  this  is  by  examining  the  preferences  for  the  Arduino  environment  as  shown  in  Figure  1.  For  example,  my  sketches  folder  on  my  Mac  is  /Users/cbell/Documents/Arduino.  Thus,  I  copied  the  folder  to  /Users/cbell/Documents/Arduino/Libraries/.

**Note**   You  need  only  one  copy  of  the  connector  in  your  Libraries  folder.  Do  **not**  place  the  MySQL\_\* files  in  your  sketch  folder  or  place  a  second  copy  elsewhere.  Doing  so  will  result  in  compilation  errors  as  the  IDE  won’t  know  which  library  files  to  use.



*Figure  1:  Arduino  Preferences  Dialog*

**Tip**         If  you  copy  a  library  to  your  Libraries  folder  while  the  Arduino  application  is  running,  you

must  restart  it  to  detect  the  new  library.

**Writing Sketches using Connector/Arduino**

Ok,  now  that  you’ve  downloaded  the  connector,  what  do  you  do  with  it?  This  section  will  explain  the  steps  needed  to  write  your  first,  simple  sketch.  We  will  also  see  some  examples  of  more  advanced  sketches  to  give  you  an  idea  of  what  is  possible.  But  first,  let’s  discuss  some  requirements  from  the  MySQL  side  of  things  then  move  on  to  the  physical  and  network  connection.  Paying  attention  here  will  save  you  tons  of  time  troubleshooting  later!

I  begin  with  the  trivial  sketch  –  simply  connecting  to  the  database  server.  I  then  present  examples  on  how  to  write  sketches  to  do  the  most  common  options.  Keep  in  mind  these  are  examples  and  that  your  specific  needs  may  require  additional  changes.

**Getting Connected Using the Ethernet Shield**

The  very  first  thing  to  do  when  setting  up  a  new  sketch  to  use  the  connector  is  to  include  the  right  libraries  and  variables.  Typically,  we  place  the  initial  calls  to  the  connector  for  startup  in  the  setup()  method.  This  includes  not  only  the  startup  code  but  also  the  call  to  connect  to  the  server.  I  present  all  of  these  in  step-­‐wise  order  starting  with  a  new  sketch.  If  you  want  to  follow  along,  open  your  Arduino  IDE  and  create  a  new,  blank  sketch.

While  this  and  the  following  examples  demonstrate  how  to  use  the  library,  keep  in  mind  there  are  several  variations  of  coding  style  and  even  choice  of  flow  that  are  also  valid  examples.  I  recommend  trying  these  as  written  before  adapting  them  to  your  own  style.

First,  add  the  required  include  files.  These  will  include  all  of  the  libraries  needed  to  compile  and  run  a  sketch  using  the  library.  Notice  we  have  included  the  Ethernet  library,  which  is  a  built-­‐in  library  that  you  do  not  need  to  download.  Next  we  include  the  MySQL\_Connector  library  in  the  form  of

including  the  header  file.  Recall  the  header  and  source  files  are  part  of  the  .zip  file  you  downloaded

that  includes  the  connector.

#include "Ethernet.h"

#include "MySQL\_Connector.h"

Next,  there  are  a  couple  of  statements  needed  to  initialize  and  work  with  the  Ethernet  class.  These  are  shown  below.  These  include  the  media  access  control  (mac1)  address  of  the  Arduino  and  the  IP  address  of  the  MySQL  server.  This  is  **not**  the  IP  address  of  the  Arduino!  The  mac  address  can  be  any  valid,  6-­‐position  hexadecimal  address  that  does  not  already  appear  on  your  network.  Thus,  you  can  use  the  one  in  this  example  but  for  projects  with  multiple  Arduino  Ethernet  nodes  you  will  want  to  make  each  mac  address  unique.

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; IPAddress server\_ip(10, 0, 1, 35);

The  next  section  defines  a  class  instance  of  the  Ethernet  client  and  the  connection  class  for  the  connector.  Here  we  must  define  the  Ethernet  client  first  as  it  is  passed  to  the  MySQL  connector  class  as  a  required  parameter.

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

**Tip**       *This  is  one  of  them  many  improvements  in  the  newest  version.  Now,  so  long  as  the  class  is  compatible  with  the  Ethernet  Client  class,  you  can  use  any  class  to  initiate  the  connector.  Which  means  you  can  use  another,  non-­‐Arduino  library  too  –  just  as  long  as  it  has  the  Ethernet  Client  as  its  ancestor.*

The  next  section  includes  the  variables  we  will  use  to  supply  the  user  credentials  for  the  connection.  In  this  case,  we  need  a  variable  to  instantiate  the  connector,  a  user  name,  and  a  password.  Be  sure  to  use  the  user  account  and  password  that  you  tested  previously.

char user[] = "root"; // MySQL user login username char password[] = "secret"; // MySQL user login password

Now  we  are  ready  to  initiate  the  Ethernet  class  and  make  the  connection  to  the  database  server.  The  following  contains  the  complete  code  to  do  this  in  the  setup()  method.  I  explain  each  line  following  the  example.

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect

Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_ip, 3306, user, password)) {

delay(1000);

// You would add your code here to run a query once on startup.

}

else

Serial.println("Connection failed.");

conn.close();

1  https://en.wikipedia.org/wiki/MAC\_address

}

The  first  line  initiates  the  serial  class.  Next,  we  initiate  the  Ethernet  class.  Here  we  pass  in  the  mac  address  we  specified  earlier.  Next,  we  issue  a  print  statement  stating  we  will  be  attempting  to  connect.  Note  that  use  of  print  statements  –  however  old  school  –  is  a  valid  way  to  track  the  progress  of  your  sketch  should  something  go  wrong.  Keep  in  mind  these  strings  you  are  printing  use  up  memory  so  make  sure  you  don’t  overuse  them,  especially  on  smaller  Arduino  boards.

The  next  construct  is  a  conditional  statement  where  we  call  the  method  to  connect  to  the  server.  In  this  case,  the  method  is  connect()  and  takes  the  following  parameters;  the  server  IP  address  (or  host  name),  server  port,  user  name,  and  password.  If  the  connection  is  successful,  the  method  will  return  a  value  that  is  evaluated  as  “true”  and  it  will  print  the  success  message.  Should  the  connection  fail,  the  method  will  return  a  value  that  is  evaluated  as  “false”  and  the  failed  message  will  print.

The  following  shows  an  example  of  the  statements  produced  in  the  serial  monitor  while  running  this

sketch.  If  you’re  following  along  with  your  own  Arduino  board,  you  should  see  something  similar.

Connecting...

Connected to server version 5.7.9-log

Query Success!

Notice  that  we  see  the  messages  from  the  sketch  indicating  a  successful  connection.  We  also  see  a  response  from  the  library  that  prints  the  version  of  the  server  to  which  it  connected.  This  can  be  helpful  in  diagnosing  failed  queries  later  on.  Listing  1  shows  the  completed  sketch  for  your  reference.  Feel  free  to  copy  it  substituting  your  specific  data  (server  address,  etc.).

*Listing  1:  Sample  Connection  Test  -­‐  Ethernet*

/\*

MySQL Connector/Arduino Example : connect

\*/

#include <Ethernet.h>

#include <MySQL\_Connection.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

char password[] = "secret"; // MySQL user login password

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

// You would add your code here to run a query once on startup.

}

else

Serial.println("Connection failed.");

conn.close();

}

void loop() {

}

Notice  the  last  line  of  code  in  our  sketch.  The  close()  method  is  used  to  disconnect  from  the  server  and  free  any  memory  used.  It  is  always  a  good  idea  to  call  this  method  to  disconnect  from  the  server  in  a  clean  manner.  If  you  plan  to  let  your  sketch  sleep  for  a  long  period  of  time,  you  can  use  the  connect()  and  close()  methods  inside  the  loop()  to  connect  only  so  long  as  you  need  to  perform  data  operations  then  disconnect.

**Getting Connected Using the WiFi Shield**

This  example  shows  the  same  operation  –  simple  connection  –  but  this  time  using  a  WiFi  shield2.  To  use  the  WiFi  shield,  you  only  need  to  change  one  small  thing  in  your  sketch.  Simply  provide  include  the  header  file  for  the  WiFi  library  and  instantiate  a  class  for  the  WiFi  client.  That’s  right  –  you  no  longer  have  to  make  changes  to  the  library  files  to  use  the  WiFi  shield.  Yippee!

#include <WiFi.h> // Use this for WiFi instead of Ethernet.h

...

WiFiClient client; // Use this for WiFi instead of EthernetClient

MySQL\_Connection conn((Client \*)&client);

Now  we  must  make  some  changes  to  our  sketch.  We  need  to  specify  two  more  variables;  the  SSID  and  password  as  follows.  This  should  match  the  settings  of  your  wireless  access  port  or  wireless router.

// WiFi card example

char ssid[] = "my\_lonely\_ssid";

char pass[] = "horse\_no\_name";

Next,  we  need  to  setup  code  to  detect  that  the  WiFi  shield  is  enabled  and  connected.  Most  of  the  time  this  isn’t  a  problem  but  if  you  turn  on  your  router  and  then  fire  up  your  Arduino,  the  WiFi  shield  may  not  have  time  to  initialize  properly.  You  could  also  use  a  delay  as  I’ve  done.  Notice  we  do  not  use  the  Ethernet.begin()  method.

Serial.begin(115200);

while (!Serial); // wait for serial port to connect. Needed for Leonardo only

// Begin WiFi section

int status = WiFi.begin(ssid, pass);

if ( status != WL\_CONNECTED) { Serial.println("Couldn't get a wifi connection");

while(true);

}

// print out info about the connection:

else {

Serial.println("Connected to network"); IPAddress ip = WiFi.localIP();

2 [https://www.arduino.cc/en/Guide/ArduinoWiFiShield](http://www.arduino.cc/en/Guide/ArduinoWiFiShield)

Serial.print("My IP address is: "); Serial.println(ip);

}

// End WiFi section

Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

}

else

Serial.println("Connection failed.");

conn.close();

Here  we  see  the  code  to  check  to  see  if  the  WiFi  is  ready  and  if  it  is,  we  retrieve  the  IP  address  assigned.  This  can  be  very  helpful  in  determining  whether  there  is  a  problem  with  subnets  between  your  Arduino  and  the  MySQL  server.  Following  this  code,  we  connect  as  usual.  A  complete  sketch  is  shown  in  listing  2  below.

*Listing  2:  Sample  Connection  Test  -­‐  WiFi*

/\*

MySQL Connector/Arduino Example : connect by wifi

\*/

#include <WiFi.h> // Use this for WiFi instead of Ethernet.h

#include <MySQL\_Connection.h>

#include <MySQL\_Cursor.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

char password[] = "secret"; // MySQL user login password

// WiFi card example

char ssid[] = "horse\_pen"; // your SSID

char pass[] = "noname"; // your SSID Password

WiFiClient client; // Use this for WiFi instead of EthernetClient

MySQL\_Connection conn((Client \*)&client);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect. Needed for Leonardo only

// Begin WiFi section

int status = WiFi.begin(ssid, pass);

if ( status != WL\_CONNECTED) { Serial.println("Couldn't get a wifi connection");

while(true);

}

// print out info about the connection:

else {

Serial.println("Connected to network"); IPAddress ip = WiFi.localIP(); Serial.print("My IP address is: ");

Serial.println(ip);

}

// End WiFi section

Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

}

else

Serial.println("Connection failed.");

conn.close();

}

void loop() {

}

If  you  have  problems  getting  your  WiFi  shield  to  work,  double  check  your  SSID  and  password  to  ensure  you  are  using  the  correct  values.  Try  these  values  on  another  computer  to  test  them.  You  should  also  refer  to  the  documentation  for  your  WiFi  shield  as  some  Arduino  compatible  WiFi  shields  require  slightly  different  startup  code.

Now  let’s  see  how  we  do  a  simple  data  collection  by  adding  an  INSERT  query.

**Basic Insert**

This  example  demonstrates  how  to  issue  a  query  to  the  database.  In  this  case,  it  is  a  simple  INSERT  that  records  the  connection  by  simply  inserting  a  row  in  a  table.  You  can  use  the  previous  example  as  a  template.  But  first,  we  need  to  create  a  test  database  and  table.  Issue  the  following  commands  on  your  MySQL  server.

CREATE DATABASE test\_arduino;

CREATE TABLE test\_arduino.hello\_arduino ( num integer primary key auto\_increment, message char(40),

recorded timestamp

);

These  commands  will  create  the  test\_arduino  database  and  a  simple  table  named  hello\_arduino  that  has  an  auto  increment  column,  a  text  string,  and  a  timestamp.  Since  the  first  and  last  columns  are  automatically  generated,  we  need  supply  only  a  text  string.

To  do  so,  we  need  to  use  an  SQL  query  such  as  the  following  INSERT  statement.

INSERT INTO test\_arduino.hello\_arduino (message) VALUES ('Hello, Arduino!');

Go  ahead  and  open  a  mysql  client,  connect  and  test  that  query.  Then  issue  a  SELECT  query  and  see  the  results.  They  should  be  similar  to  the  following.  If  you  run  the  command  several  times,  you  will  see  multiple  rows  in  the  result  set.

mysql> SELECT \* FROM test\_arduino.hello\_arduino;

+-----+-----------------+---------------------+

| num | message | recorded |

+-----+-----------------+---------------------+

| 1 | Hello, Arduino! | 2015-07-27 14:39:13 |

+-----+-----------------+---------------------+

1 row in set (0.00 sec)

As  you  can  see,  each  time  we  insert  this  data  we  will  get  a  new  row  in  the  table  complete  with  a  unique  key  (auto  generated)  and  a  timestamp  of  when  the  row  was  inserted.  Cool!  Now,  let’s  add  this  to  our  sketch.

To  do  so,  we  add  a  new  string  variable  to  contain  the  query  then  use  the  MySQL\_Cursor  class  to  execute  the  query.  To  use  the  cursor,  we  add  another  include  directive  to  include  the  cursor  header  file,  then  dynamically  allocate  the  object  with  a  new  operation,  perform  the  query,  then  use  a  delete  operation  to  free  the  object  and  all  of  its  memory.  The  following  shows  the  steps  in  order.

#include <MySQL\_Cursor.h>

...

MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);

cur\_mem->execute(INSERT\_SQL);

delete cur\_mem;

Notice  call  the  execute()  method  to  run  the  query.  Listing  3  shows  the  completed  sketch.  Notice  we  only  added  three  lines  of  code  and  changed  the  one  print  statement  to  clarify  the  flow  (shown  in  bold).

*Listing  3:  Simple  Data  Insert  Sketch*

/\*

MySQL Connector/Arduino Example : basic insert

\*/

#include <Ethernet.h>

#include <MySQL\_Connection.h>

#include <MySQL\_Cursor.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

char password[] = "secret"; // MySQL user login password

// Sample query

**char INSERT\_SQL[] = "INSERT INTO test\_arduino.hello\_arduino (message) VALUES ('Hello, Arduino!')";**

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

}

else

Serial.println("Connection failed.");

}

void loop() {

delay(2000);

Serial.println("Recording data.");

**// Initiate the query class instance**

**MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);**

**// Execute the query**

**cur\_mem->execute(INSERT\_SQL);**

**// Note: since there are no results, we do not need to read any data**

**// Deleting the cursor also frees up memory used delete cur\_mem;**

}

Notice  also  we  put  the  code  to  run  the  query  in  the  loop()  method,  which  means  it  will  execute  repeatedly  until  power  down  your  Arduino.  This  is  because  larger,  more  meaningful  sketches  that  insert  data  periodically  the  data  recording  code  would  be  put  in  the  loop()  method.

Go  ahead  and  run  this  several  times  then  issue  the  SELECT  query  again.  You  should  now  see  one  row

for  each  time  the  sketch  ran  (plus  how  ever  many  tests  you  did  previously).    Now  let’s  see  a  more  complex  data  insert  with  variables.

**Complex Insert**

The  most  frequent  use  of  the  connector  is  recording  data  collected  by  the  Arduino.  This  could  be  a  sensor  (or  several)  such  as  temperature,  latch  occurrence  (door  open/closed),  button  pressed,  etc.  As  such,  we  only  need  to  record  the  data  and  move  on.  However,  the  data  in  this  case  is  likely  to  be  something  read  or  generated  rather  than  a  static  string.

To  insert  data  that  is  generated  (or  read),  one  must  build  the  query  string  before  issuing  the  query.  We  do  this  using  the  sprintf()method.  The  following  example  simulates  reading  a  sensor.  The  query  is  still  inside  the  setup()  method  as  we  only  want  to  do  this  once  as  a  test.

Before  we  being,  let’s  create  a  new  table  that  will  store  the  results  of  an  integer  and  float  value  read.  We  will  also  keep  the  text  string  to  label  the  observation  –  in  this  case  a  simulated  sensor  node.  Since  most  sensors  produce  floating-­‐point  numbers,  I  include  one  field  to  demonstrate  how  to  convert  floating-­‐point  numbers.

CREATE TABLE test\_arduino.hello\_sensor ( num integer primary key auto\_increment, message char(40),

sensor\_num integer, value float, recorded timestamp

);

The  following  is  called  a  format  string  used  by  the  sprintf()  method  to  form  the  string.  This  works  by  substituting  values  from  variables  for  the  special  characters  in  the  format  string  itself.  As  you  may  surmise,  we  will  be  building  a  new  string  and  thus  will  be  allocating  more  memory  for  this.  As  I  eluded  to  earlier,  the  more  of  these  special  strings  you  must  build,  the  more  memory  you  are  likely  to  consume  and  thus  if  using  a  smaller  Arduino  board  you  must  be  miserly  with  your  variables.  The  following  is  the  format  string  for  this  example.

INSERT INTO test\_arduino.hello\_sensor (message, sensor\_num, value) VALUES ('%s',%d,%s)

Notice  we  have  three  variables  here.  The  first,  a  message,  is  just  a  string  we  pass.  The  second  is  the

sensor  number  (and  integer).  The  last  is  a  floating-­‐point  number.  While  we  use  a  %s  to  signify  a

string  and  a  %d  to  signify  the  integer  for  substitution,  we  have  another  string  %s  for  the  floating-­‐point  value.  This  is  because  the  Arduino  library  does  not  currently  support  converting  floating-­‐point  numbers  in  sprintf().  Thus,  we  must  use  the  dtostrf()3  method  as  illustrated  in  the  code  snippet  below.

dtostrf(50.125, 1, 1, temperature);

sprintf(query, INSERT\_DATA, "test sensor", 24, temperature);

conn.execute(query);

Here  we  are  converting  the  floating  point  value  50.125  to  a  string  and  storing  it  a  variable  named  temperature,  which  we  later  use  in  the  sprintf()  call  along  with  our  message  (test  sensor)  and  sensor  number  (24).  Thus,  keep  in  mind  that  floating-­‐point  numbers  are  a  bit  messy  to  deal  with.  The  good  news  is  this  code  works  really  well.  You  should  end  up  with  a  result  similar  to  the  output  below.

mysql> select \* from test\_arduino.hello\_sensor;

+-----+-------------+------------+-------+---------------------+

| num | message | sensor\_num | value | recorded |

+-----+-------------+------------+-------+---------------------+

| 1 | test sensor | 24 | 50.1 | 2015-07-27 15:12:38 |

+-----+-------------+------------+-------+---------------------+

1 row in set (0.00 sec)

The  variables  we  need  for  this  sketch  include  one  for  a  buffer  to  store  the  formatted  query,  the  query  format  string,  and  a  buffer  for  the  temperature.  Listing  4  shows  the  complete  sketch  with  the  new  statements  in  bold.

*Listing  4:  Complex  Insert  Sketch*

/\*

MySQL Connector/Arduino Example : complex insert

\*/

#include <Ethernet.h>

#include <MySQL\_Connection.h>

#include <MySQL\_Cursor.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

3 [http://www.atmel.com/webdoc/AVRLibcReferenceManual/index.htm](http://www.atmel.com/webdoc/AVRLibcReferenceManual/index.html)l (Search  for  dtostrf)

char password[] = "secret"; // MySQL user login password

// Sample query

**char INSERT\_DATA[] = "INSERT INTO test\_arduino.hello\_sensor (message, sensor\_num, value) VALUES ('%s',%d,%s)";**

**char query[128];**

**char temperature[10];**

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

**// Initiate the query class instance**

**MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);**

**// Save**

**dtostrf(50.125, 1, 1, temperature);**

**sprintf(query, INSERT\_DATA, "test sensor", 24, temperature);**

**// Execute the query cur\_mem->execute(query);**

**// Note: since there are no results, we do not need to read any data**

**// Deleting the cursor also frees up memory used delete cur\_mem;**

Serial.println("Data recorded.");

}

else

Serial.println("Connection failed.");

conn.close();

}

void loop() {

}

You  could  make  the  buffers  dynamic  –  and  that  would  probably  be  a  good  idea  –  just  make  sure  you  always  release  the  memory  after  you’re  done  otherwise  you  will  run  out  of  memory  quickly.  Also,  make  sure  the  variables  or  memory  allocated  is  large  enough  to  store  the  formatted  strings.  The  sprintf()  method  will  not  fail  and  instead  will  overflow  the  memory  which  can  cause  all  manner  of  pain  so  be  sure  to  double  check  your  memory  allocation  (static  or  dynamic)!

As  you  can  see,  the  hardest  part  of  collecting  data  is  managing  the  buffers  needed  for  the  queries  and  making  sure  you  release  any  allocated  memory.  You  may  think  selecting  data  would  be  a  bit  easier,  and  it  is,  but  it  requires  a  bit  more  work  to  make  use  of  the  data  returned.

**Basic Select**

Sometimes  it  is  necessary  to  retrieve  data  from  your  database  for  use  in  your  sketch.  Whether  you

are  reading  from  a  table  of  values  or  reading  the  results  of  another  Arduino  project,  the  data  required

will  likely  be  used  in  some  form  of  calculation.

This  example  shows  a  simple  SELECT  query  that  retrieves  one  row  from  the  database  and  stores  it  in  a  variable  for  use  in  the  sketch.  Like  the  other  examples,  I’ve  made  this  as  simple  as  possible  by  placing  the  code  in  the  setup()  method.

Before  we  begin,  let  us  consider  what  is  happening  here.  First,  we  are  issuing  a  SELECT  statement  to  the  database,  which  will  return  one  or  more  rows  (depending  on  the  query).  But  before  that,  the  database  will  return  a  list  of  columns  and  following  that  one  row  at  a  time  until  no  more  rows  are  left.  Thus,  we  must  first  read  the  columns  then  one  row  at  a  time.  Let’s  see  how  to  do  this  in  the  following  example  starting  with  the  query.

**Note**         *This  query  is  issued  against  the  world  sample  database.  You  can  download  this  database  from  the  following  link*[*(http://dev.mysql.com/doc/inde*](http://dev.mysql.com/doc/index-)*x- other.html).  To  run  this  sketch,  you  will  need  to  download  the  file,  unzip  it,  and  follow  the  instructions  on  the  website  to  install  it.  Once  you  install  the  world  database,  you  can  run  the  query  in  a  mysql  client.*

mysql> SELECT population FROM world.city WHERE name = 'New York';

+------------+

| population |

+------------+

| 8008278 |

+------------+

1 row in set (0.01 sec)

Notice  there  is  one  row  returned.  The  following  is  the  code  we  need  to  read  this  value.

MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);

// Execute the query cur\_mem->execute(query);

// Fetch the columns (required) but we don't use them. column\_names \*columns = cur\_mem->get\_columns();

// Read the row (we are only expecting the one)

do {

row = cur\_mem->get\_next\_row();

if (row != NULL) {

head\_count = atol(row->values[0]);

}

} while (row != NULL);

// Deleting the cursor also frees up memory used delete cur\_mem;

// Show the result Serial.print(" NYC pop = "); Serial.println(head\_count);

Notice  we  first  execute  the  query  then  read  the  columns.  This  is  a  special  function  in  the  library.  If  you  want  to  read  the  column  names,  you  can  but  that  is  a  rarely  used  operation.  We  will  see  how  to  do  this  in  the  next  example.

Next,  we  read  the  rows  one  at  a  time.  Since  we  know  the  query  returns  only  one  row,  you  may  be  tempted  to  code  only  the  one  get\_next\_row()  call,  but  do  not  do  this.  While  we  only  see  the  one  row  in  the  result,  there  is  an  acknowledgement  or  trailing  packet  after  the  last  row  read  and  thus  you  must  code  the  loop  even  if  there  is  only  one  row  returned.

Once  the  row  is  read,  we  use  the  atol()  method  to  save  the  value  read  from  the  row  from  the  first  column  (starts  counting  at  0).  You  can  use  the  row  variable  to  reference  any  column  you  need  if  the  row  returns  more  than  a  single  column.  But  be  careful  because  the  more  columns  returned,  the  more  memory  will  be  consumed.  That  is  why  we  specified  the  one  column  in  the  query  –  to  save  space  and  request  only  the  data  needed  and  nothing  more.  You  should  adopt  this  misery  practice  when  writing  sketches.

Finally,  we  print  out  the  results  we  read  from  the  row.  Listing  5  shows  the  completed  sketch.  Try  it  yourself  to  ensure  you  get  the  same  value  from  the  database.  Once  again,  the  new  lines  of  code  are  highlighted  in  bold.

*Listing  5:  Simple  Select  Sketch*

/\*

MySQL Connector/Arduino Example : basic select

\*/

#include <Ethernet.h>

#include <MySQL\_Connection.h>

#include <MySQL\_Cursor.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

char password[] = "secret"; // MySQL user login password

// Sample query

**char query[] = "SELECT population FROM world.city WHERE name = 'New York'";**

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

// Create an instance of the cursor passing in the connection

MySQL\_Cursor cur = MySQL\_Cursor(&conn);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

}

else

Serial.println("Connection failed.");

}

void loop() {

**row\_values \*row = NULL;**

**long head\_count = 0;**

delay(1000);

Serial.println("1) Demonstrating using a cursor dynamically allocated.");

**// Initiate the query class instance**

**MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);**

**// Execute the query cur\_mem->execute(query);**

**// Fetch the columns (required) but we don't use them.**

**column\_names \*columns = cur\_mem->get\_columns();**

**// Read the row (we are only expecting the one)**

**do {**

**row = cur\_mem->get\_next\_row();**

**if (row != NULL) {**

**head\_count = atol(row->values[0]);**

**}**

**} while (row != NULL);**

**// Deleting the cursor also frees up memory used delete cur\_mem;**

**// Show the result Serial.print(" NYC pop = "); Serial.println(head\_count);**

}

The  next  example  combines  the  need  to  pass  in  variables  to  the  SELECT  query  to  retrieve  data  based

on  dynamic  information.

**Complex Select**

This  example  shows  how  to  use  a  SELECT  query  with  a  WHERE  clause  formed  from  a  calculation.  In  this  case,  we  simulate  the  calculation  with  the  use  of  an  arbitrary  number.  However,  you  can  simply  replace  that  logic  with  the  reading  from  user  input,  a  sensor,  another  Arduino,  calculations  in  your  sketch,  etc.

We  still  use  the  world  database  but  in  this  case  we  want  to  select  those  countries  with  a  specific

population  (i.e.,  greater  than  a  specific  value  provided).  The  query  we  want  to  use  is  the  following.

SELECT name, population FROM world.city

WHERE population > 9000000

ORDER BY population DESC;

There  is  a  lot  going  on  here!  Notice  not  only  do  we  specify  the  population,  we  also  sort  the  result  by  population.  We  will  therefore  see  how  to  navigate  a  multiple  row  result  set  as  well  as  see  how  to  print  the  column  names  returned.

Notice  the  size  of  the  variable  we  want  to  set  for  the  WHERE  clause.  Here  we  will  use  another  sprintf()call  to  format  the  string.  In  this  case,  we  need  a  long  integer  thus  we  use  %lu  (unsigned  long).

Listing  6  shows  the  code  needed  to  read  the  columns,  print  them  out,  then  read  the  rows  and  display

the  values  with  the  pertinent  code  in  bold.

*Listing  6:  Complex  Select  Sketch*

/\*

MySQL Connector/Arduino Example : complex select

\*/

#include <Ethernet.h>

#include <MySQL\_Connection.h>

#include <MySQL\_Cursor.h>

byte mac\_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server\_addr(10,0,1,35); // IP of the MySQL \*server\* here char user[] = "root"; // MySQL user login username

char password[] = "secret"; // MySQL user login password

**// Sample query**

**//**

**// Notice the "%lu" - that's a placeholder for the parameter we will**

**// supply. See sprintf() documentation for more formatting specifier**

**// options**

**const char QUERY\_POP[] = "SELECT name, population FROM world.city WHERE**

**population > %lu ORDER BY population DESC;";**

**char query[128];**

EthernetClient client;

MySQL\_Connection conn((Client \*)&client);

void setup() { Serial.begin(115200);

while (!Serial); // wait for serial port to connect Ethernet.begin(mac\_addr); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(1000);

}

else

Serial.println("Connection failed.");

}

void loop() {

delay(1000);

Serial.println("> Running SELECT with dynamically supplied parameter");

**// Initiate the query class instance**

**MySQL\_Cursor \*cur\_mem = new MySQL\_Cursor(&conn);**

**// Supply the parameter for the query**

**// Here we use the QUERY\_POP as the format string and query as the**

**// destination. This uses twice the memory so another option would be**

**// to allocate one buffer for all formatted queries or allocate the**

**// memory as needed (just make sure you allocate enough memory and**

**// free it when you're done!). sprintf(query, QUERY\_POP, 9000000);**

**// Execute the query**

**cur\_mem->execute(query);**

**// Fetch the columns and print them column\_names \*cols = cur\_mem->get\_columns();**

**for (int f = 0; f < cols->num\_fields; f++) { Serial.print(cols->fields[f]->name);**

**if (f < cols->num\_fields-1) {**

**Serial.print(',');**

**}**

**}**

**Serial.println();**

**// Read the rows and print them row\_values \*row = NULL;**

**do {**

**row = cur\_mem->get\_next\_row();**

**if (row != NULL) {**

**for (int f = 0; f < cols->num\_fields; f++) { Serial.print(row->values[f]);**

**if (f < cols->num\_fields-1) {**

**Serial.print(',');**

**}**

**}**

**Serial.println();**

**}**

}

**} while (row != NULL);**

**// Deleting the cursor also frees up memory used delete cur\_mem;**

Notice  how  the  code  is  written  to  loop  over  the  columns  first  then  the  values  for  each  row.  Now  that  we’ve  seen  some  examples  of  sketches  and  common  uses  of  the  library,  the  next  section  discusses  some  tips  and  techniques  for  writing  your  sketches  to  interact  with  MySQL.

**Tips for Writing Sketches with the Connector**

This  section  contains  a  list  of  suggestions  for  making  better  sketches  with  the  connector.  In  some  cases  this  is  advice  and  in  other  cases  it  is  suggested  code  or  techniques.  If  your  sketch  will  include  more  complex  queries  than  those  shown  above,  you  should  read  this  section  for  incorporation  into  your  own  sketches.

**Use the Examples**

There  are  many  example  sketches  included  with  the  connector.  You  should  run  one  or  more  of  these  to  ensure  you  understand  how  the  connector  works  before  writing  your  own  sketch.  I  recommend  starting  with  the  connect,  basic\_insert,  and  basic\_select  examples  first.  Get  to  know  these  and  test  them  to  ensure  your  MySQL  server  is  setup  correctly  and  your  Arduino  can  connect  to  it.  If  you  have  trouble  with  these  examples,  do  not  blame  the  connector  (at  least  not  initially).  Read  the  troubleshooting  section  below  to  solve  one  or  more  of  the  common  problems  and  try  your  example  again.  Don’t  forget  to  change  the  IP  address,  user  name  and  password!

**Keep It Simple**

This  one  I  feel  is  a  given  for  writing  code  for  microprocessors,  but  you  may  be  surprised  at  the  number  of  requests  I’ve  had  for  helping  solve  problems.  The  root  cause  or  the  significant  factor  for  much  of  the  users’  trouble  stems  around  making  the  sketch  far  more  complex  than  it  needs  to  be.

This  is  especially  true  for  those  that  write  their  entire  solution  before  testing  it.  That  is,  they  write  hundreds  of  lines  of  code,  get  it  to  compile  (sometimes  not  so  much)  then  try  to  run  it.  In  this  case,  the  user  has  failed  to  realize  all  aspects  of  their  solution  should  be  tested  in  a  step-­‐wise  fashion.

For  example,  write  the  sketch  to  do  the  minimalist  steps  needed  to  demonstrate  (test)  each  part.  For  working  with  the  MySQL  database,  start  with  a  simple  connection  test  then  proceed  to  testing  each  and  every  query  using  dummy  data  or  simulated  values.

Likewise,  working  with  sensors  or  other  devices  should  be  done  in  isolation  so  that  you  can  eliminate

major  portions  of  the  sketch  for  investigation  should  something  go  wrong.

If  you  adopt  this  philosophy,  your  sketches  will  be  easier  to  write  and  you  will  have  far  more  success

than  the  “code  it  once  and  pray  it  works”  philosophy.

**Connect/Close**

Most  sketches  are  written  to  connect  once  at  startup.  However,  for  complex  solutions  that  collect  or  interact  with  the  database,  the  connection  is  critical  for  longer  running  projects.  It  is  often  the  case  that  networks  can  become  unreliable.  Indeed,  there  is  nothing  in  the  specification  of  the  networking  protocols  or  equipment  to  suggest  it  is  always  lossless.  In  fact,  the  network  is  design  to  be  “mostly”  reliable  with  some  acceptable  loss.

When  loss  occurs,  it  can  sometimes  cause  errors  in  the  connector  when  reading  from  the  database  or  can  cause  the  Ethernet  shield  to  drop  its  connection.  In  extreme  cases,  it  can  cause  the  sketch  to  hang  or  loop  out  of  control  (depending  on  how  the  conditional  statements  are  written).

To  combat  this,  we  can  use  a  technique  whereby  we  connect  and  close  on  each  pass  through  the  loop.  This  will  work,  but  there  is  a  more  elegant  solution  that  allows  you  to  reconnect  whenever  the  connection  is  dropped.  The  following  demonstrates  this  concept.

void loop() {

delay(1000);

if (conn.connected()) {

// do something

} else { conn.close(); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(500);

Serial.println("Successful reconnect!");

} else {

Serial.println("Cannot reconnect! Drat.");

}

}

}

Notice  here  we  check  the  status  of  the  connector  and  if  it  is  not  connected,  we  reconnect.  This  will

save  you  from  cases  where  the  connection  is  dropped  to  network  or  database  errors.

**Reboot Fix**

Closely  related  to  the  connect/close  technique  is  a  technique  to  reboot  the  Arduino  should  something  bad  happen.  This  can  be  really  handy  if  you  have  a  project  that  must  work  but  is  Ok  if  there  are  short  data  gaps.  For  example,  if  you  are  monitoring  something  and  performing  calculations  it  is  possible  your  hardware  could  have  periodic  issues  as  well  as  logic  errors  or  simple  networking  failures.

To  overcome  these  situations,  you  can  program  the  Arduino  to  reboot  using  the  following  code.  Note  that  this  shows  this  technique  used  with  the  connect/close  option  as  they  are  complimentary.  After  all,  if  you  cannot  connect  after  N  tries,  a  reboot  cannot  hurt  and  in  most  cases  where  it  is  a  problem  with  memory  or  the  Ethernet  shield  or  related,  it  works.

void soft\_reset() {

asm volatile("jmp 0");

}

void loop() {

delay(1000);

if (conn.connected()) {

// do something num\_fails = 0;

} else { conn.close(); Serial.println("Connecting...");

if (conn.connect(server\_addr, 3306, user, password)) {

delay(500);

Serial.println("Successful reconnect!");

num\_fails++;

if (num\_fails == MAX\_FAILED\_CONNECTS) {

Serial.println("Ok, that's it. I'm outta here. Rebooting...");

delay(2000);

soft\_reset();

}

}

}

}

Notice  here  we  use  an  assembler  call  to  jump  to  position  0.  This  effectively  reboots  the  Arduino  microcode.  Cool,  eh?    And  you  thought  you’d  have  to  slog  out  to  the  pig  farm  and  press  the  wee  little  reboot  button.

**Memory Checker**

Another  useful  technique  is  monitoring  or  diagnosing  memory  problems  by  calculating  how  much

memory  is  remaining.  We  do  this  with  the  following  method.

int get\_free\_memory()

{

extern char bss\_end; extern char \* brkval; int free\_memory; if((int) brkval == 0)

free\_memory = ((int)&free\_memory) - ((int)& bss\_end);

else

free\_memory = ((int)&free\_memory) - ((int) brkval);

return free\_memory;

}

You  can  use  this  method  anywhere  in  the  code.  I  like  to  use  it  with  a  print  statement  to  print  out  the

value  calculated  as  follows.

Serial.print(" RAM: "); Serial.println(get\_free\_memory());

Placing  this  code  strategically  in  the  sketch  and  watching  the  results  in  the  serial  monitor  can  help

you  spot  memory  leaks  and  situations  where  you  run  out  of  memory.

**Do Your Homework!**

It  is  at  this  point  that  I  would  like  to  clarify  one  thing  about  using  libraries  such  as  the  connector.  This  is  advice  for  all  who  are  learning  how  to  program  your  Arduino.  Be  sure  to  do  your  homework  and  your  own  research  before  asking  questions.  So  many  times  I  get  questions  about  the  most  basic

things  (well,  basic  to  the  experienced)  that  have  nothing  to  do  with  the  connector.  For  example,

working  with  memory,  variables,  and  strings  seem  to  be  stumbling  blocks  for  new  users.

In  the  end,  you  will  get  far  more  useful  help  from  library  authors  and  other  experienced  Arduinistas

if  you  take  some  time  to  read  a  book,  web  page,  or  listen  to  a  podcast  before  contacting  the  author  for  help  or  complain  about  a  compiler  error.  A  small  amount  of  learning  on  your  part  will  reap  dividends  when  you  can  ask  a  specific  question  or  seek  help  for  a  complex  issue.

A  case  in  point  is  this  document.  From  my  experience,  this  document  is  far  more  detailed  than  any  other  library  available  for  the  Arduino  (with  notable  exceptions).  Part  of  the  motivation  for  writing  this  document  was  to  consolidate  the  information  about  the  connector  and  to  ensure  those  new  to  using  the  connector  had  a  sufficiently  detailed  tutorial.  The  following  section  completes  the  body  of  information  about  the  connector  by  presenting  the  most  common  questions  asked  of  users.

**Troubleshooting**

This  section  presents  a  short  but  proven  practice  for  troubleshooting  sketches  that  use  the  connector.  Should  you  have  a  situation  where  your  sketch  fails  or  doesn’t  work  when  modified  or  moved  to  another  network,  deployed,  etc.,  following  this  process  can  help  isolate  the  problem.

1. Verify  the  network.  Try  connecting  another  computer  in  place  of  the  Arduino  to  ensure  you  can  connect  to  the  network  and  the  database  sever.  Correct  any  network  issues  before  moving  on.

2. Verify  your  user  account.  With  the  same  computer,  try  logging  into  the  database  using  the  credentials  in  your  sketch.  Correct  any  issues  with  permissions  and  user  accounts  before  moving  on.

3. Check  permissions.  If  you  restart  your  Arduino  and  still  cannot  connect,  go  back  and  verify

your  permissions  again.

4. Check  network  hardware.  Make  sure  there  are  no  firewalls,  port  scanners,  etc.  that  are

blocking  access  to  the  database  server.

5. Isolate  your  code.  Once  all  connection  problems  are  solved,  check  your  code.  Chances  are

you  can  comment  out  or  remove  most  of  the  code  to  check  only  the  bare  minimum  parts.  I

recommend  breaking  the  code  into  sections  and  testing  each  until  you  encounter  the  section

with  the  problem.

6. Check  your  hardware.  When  all  else  fails,  try  another  Arduino.  I’ve  seen  cases  where  an  Arduino  breaks  or  has  a  short  or  some  other  failure  where  it  can  boot  and  run  simple  sketches  but  anything  more  than  that  it  fails.

**Frequently Asked Questions**

The  following  are  a  list  of  questions  that  have  been  asked  numerous  times  on  the  forums.  They  address  a  lot  of  common  pitfalls  and  explain  a  few  new  techniques  not  discussed  above.  Be  sure  to  scan  this  list  before  making  new  inquiries  on  the  forums.  The  following  are  listed  in  no  particular  order.

***Can  I  use  the  connector  to  connect  to  other  database  servers?***

No.  The  connector  only  works  with  MySQL  server.

***Can  I  use  the  connector  with  non-­‐Arduino  compatible  Ethernet  modules?***

The  connector  only  works  with  Ethernet  shields  and  modules  that  support  the  Arduino  Ethernet

class.  If  your  module  requires  a  new  Ethernet  class,  it  will  not  work  with  the  connector.

***My  sketch  is  locking  up.  What  do  I  do?***

The  problem  can  be  one  of  several  things,  but  the  most  likely  cause  is  running  out  of  memory  or  dropping  the  network.  Check  your  memory  usage  to  ensure  you  have  enough  memory.  You  can  switch  to  a  large  Arduino  if  your  sketch  outgrows  your  board.  For  network  issues,  you  can  use  the  connect/close  or  the  reboot  techniques  above.

***I’m  getting  “multiple  definition  of  `Connector::check\_ok\_packet()'”  and  similar  compiler  errors.  What’s  wrong?***

If  you  are  seeing  compiler  errors  about  duplicate  functions  and  similar,  it  is  because  you  have  the  connector  code  in  more  than  one  place.  That  is,  you  have  duplicated  the  code  in  your  Libraries  or  sketch  folder.  On  Windows  machines,  this  is  possible  if  you  copy  the  archive  to  a  temporary  folder  or  unzip  it  in  multiple  locations.  Be  sure  only  one  copy  of  the  mysql.\* files  exist  in  the  Libraries  folder.

***I’m  getting  “error:  'column\_names'  was  not  declared  in  this  scope”  and  similar  compiler  errors.  What’s  wrong?***

You  must  enable  WITH\_SELECT  in  mysql.h  to  enable  the  methods  for  processing  result  sets  (SELECT  queries).

***I’m  getting  compiler  errors  in  the  SHA1  libraries.  What’s  wrong?***

If  you  used  an  older  version  of  the  connector  and  upgraded  recently,  it  is  possible  your  SHA1  folder  is

out  of  date.  Be  sure  to  copy  the  latest  sha1  folder  from  the  .zip  archive  and  restart  your  IDE.

***I  get  compiler  errors  when  trying  to  do  a  query  with  variables.***

Be  sure  to  sprintf()  and  dtostrf()  to  format  your  query  with  variables.  The  code  does  not

support  variable  substitution.

***What  is  the  3306  in  the  example  code?***

It  is  the  port  on  which  the  MySQL  server  is  listening.  You  can  specify  another  port,  but  your  MySQL

server  must  be  setup  to  listen  on  the  port.  3306  is  the  default  setting.

***My  queries  aren’t  working!***

You  should  test  your  queries  using  the  mysql  client  before  attempting  to  run  your  sketch.  Many  times

there  are  small  syntax  errors  that  you  must  fix  before  the  query  will  work.

***Why  aren’t  select  queries  enabled  by  default?***

I  purposefully  disabled  the  code  to  process  result  sets  to  save  a  few  bytes.  That  is,  if  your  sketch  (like

most)  are  just  inserting  data,  it  does  not  need  the  extra  code  taking  up  valuable  memory.

***I  keep  getting  “Connection  failed”.***

If  you  are  getting  a  connection  failed  message  (as  written  into  your  sketch),  it  is  most  likely  your  Arduino  is  not  connected  to  the  network  properly  or  your  user  account  and  password  is  not  correct  or  the  user  does  not  have  permissions  to  connect.  Use  a  second  computer  and  the  credentials  from  your  sketch  to  check  to  see  you  can  connect.  Resolve  any  issues  and  retry  your  sketch.

***I  get  the  error,  “Connector  does  not  name  a  type”.  What’s  wrong?***

The  most  likely  scenario  is  you  have  not  placed  the  connector  in  your  Arduino  Libraries  folder  or  you  have  renamed  it  or  you  placed  it  in  another  folder.  Be  sure  it  is  installed  correctly  and  restart  your  IDE.

***Can  I  assign  an  IP  to  the  Arduino?***

Yes,  use  one  of  the  alternative  set  of  parameters  for  the  Ethernet  class  to  setup  the  IP  manually.  See

[https://www.arduino.cc/en/Reference/EthernetBegin.](http://www.arduino.cc/en/Reference/EthernetBegin)

***Can  I  use  a  hostname  instead  of  an  IP  address  for  the  server?***

Yes,  but  it  requires  using  the  dns  library  as  follows.

#include <Dns.h>

…

char hostname[] = ["www.google.com](http://www.google.com)”; // change to your server’s hostname/URL

…

IPAddress server\_ip; DNSClient dns; dns.begin(Ethernet.dnsServerIP()); dns.getHostByName(hostname, server\_ip); Serial.println(server\_ip) Serial.println("Connecting...");

if (conn.connect(server\_ip, 3306, user, password)) {

…

***Can  I  use  more  than  one  query?***

Yes,  just  make  sure  you  have  enough  memory  for  the  strings.

***I  get  PACKET\_ERROR.  What’s  that?***

This  error  occurs  when  the  connector  receives  the  wrong  packet  header  or  an  unexpected  response  from  the  server.  It  occurs  most  often  when  using  select  queries  where  there  are  additional  rows  that  are  not  read.  See  the  examples  above  to  ensure  you  are  processing  the  entire  result  set.  You  can  also  use  a  WHERE  or  LIMIT  clause  to  help  restrict  the  number  of  rows  returned.

***I  see  garbage  characters  in  the  serial  monitor.***

Check  to  make  sure  the  baud  rate  of  the  serial  monitor  matches  your  sketch.  Change  one  or  the  other

to  match  and  you  should  see  valid  characters.

***I  get  Connection  Failed.  What  could  be  wrong?***

You  have  one  or  more  of  the  following  incorrect:

• server  address

• using  static  IP  (try  DHCP)

• the  network  connection  isn't  viable  or  behind  a  switch

• the  user  credentials  do  not  work

Your  best  diagnostic  is  to  use  a  second  computer  on  the  same  Ethernet  line  with  the  same  credentials  (server  address,  user,  password)  and  attempt  to  connect.  If  you  can,  then  you  may  have  a  problem  with  your  hardware.

***I  still  cannot  get  the  connection  to  work,  what  else  can  I  try?***

You  should  use  one  of  the  examples  that  come  with  the  Arduino  IDE  such  as  the  Web  Client  sketch.  Try  this  and  if  that  works,  you  know  your  Ethernet  shield  is  working.  You  can  do  the  same  for  the  WiFi  shield.  Once  you  verify  the  shield  works,  go  back  and  check  your  MySQL  server  and  test  connecting  to  it  from  another  computer  until  the  credentials  and  permissions  are  correct.

***I  am  using  a  second  computer  but  I  still  cannot  login  to  the  database.***

The  top  causes  are:

• the  IP  address  of  the  server  has  changed

• there  is  a  firewall  blocking  incoming  connections  on  3306

• the  network  port/router/switch  doesn't  work

• the  user  and  host  permissions  are  not  correct  (Cannot  login)

***How  can  I  find  my  MySQL  server  IP  address?***

There  are  many  ways.  If  you  are  running  Linux,  Unix,  or  Mac  OS  X,  use  this:

ifconfig

For  Windows  use  this:

ipconfig

You  will  find  the  IP  address  in  the  output  of  these  commands.  You  can  also  do  this  in  a  mysql  client:

show variables like 'hostname';

Then  use  ping  (from  a  terminal)  to  ping  the  hostname  shown.  The  output  will  show  the  IP  address.

***Does  the  connector  work  with  GPRS  modules?***No.  Only  the  Arduino  Ethernet  or  WiFi  shields.  ***How  do  I  record  the  date  and  time  of  my  event?***

Use  a  timestamp  column  in  your  table.  This  will  be  updated  with  the  current  time  and  date  when  the

row  is  inserted.

***How  do  I  use  PROGMEM  for  storing  strings?***

Include  the  program  memory  space  header  then  declare  your  string  with  the  keyword  as  shown.  Remember  to  use  the  optional  second  parameter  in  the  execute()  method  when  passing  in  these  strings  for  queries.

#include <avr/pgmspace.h>

…

const PROGMEM char query[] = "SELECT name, population FROM world.city";

…

conn.execute(query, true);

***Can  I  use  the  new  WiFi  Shield  101?***

Yes.  There  is  an  example  on  how  to  use  the  new  WiFi  Shield  101.  See  the  *File-­‐>Examples-­‐>MySQL  Connector  Arduino*menu.

***Cam  I  use  the  Ethernet  Shield  2?***

Yes  and  no.  Yes,  the  connector  will  work  with  the  new  shield  but  you  will  need  to  make  a  minor

change  to  the  MySQL\_Packet.h file.  Open  the  MySQL\_Packet.h  file  and  change:

#include <Ethernet.h>

 to:

#include <Ethernet2.h>

And  no  because  you  cannot  use  the  new  shield  (currently)  with  the  Arduino  IDE  from  arduino.cc.  You  must  download  the  arduino.org  software,  not  the  software  from  arduino.cc.  Yes,  there  is  a  difference.  I  won't  go  into  that  here,  but  suffice  to  say  there  are  differences.  To  download  the  IDE,  go  to [http://www.arduino.org/softwar](http://www.arduino.org/software)e.  You  can  run  it  along  side  another  version,  just  make  sure  you  install  it  in  another  location.  Once  installed,  you  can  compile  your  sketch  but  first  change  the  include  directives  to  list  the  following.

#include <SPI.h> // <---- Add this include

#include <Ethernet2.h> // <---- Change to use the new library :)

**Limitations**

Given  the  target  platform  –  a  small  microcontroller  with  limited  memory  -­‐  there  are  some  limitations  to  using  a  complex  library  on  the  Arduino  platform.  The  first  thing  you  should  know  about  the  connector  is  it  isn't  a  small  library  and  can  consume  a  lot  of  memory.  While  the  library  uses  dynamic  memory  to  keep  memory  use  to  a  minimum,  how  much  memory  is  used  depends  on  how  you  use  the  connector.

More  specifically,  you  will  need  to  limit  how  many  string  constants  you  create.  If  you  are  issuing  simple  data  insertion  commands  (INSERT INTO),  an  easy  way  to  calculate  this  is  the  connector  uses  a  bit  more  than  the  maximum  the  size  of  the  longest  query  string  in  addition  to  the  sum  of  all  of  your  strings.  If  you  are  querying  the  server  for  data,  the  connector  uses  a  bit  more  than  the  cumulative  size  of  a  row  of  data  returned.

There  are  other  limitations  to  consider  but  most  notable  is  memory  usage.  If  you  are  using  the  latest  Arduino  Due  this  may  not  be  an  issue.  But  there  are  other  considerations.  The  following  lists  the  known  limitations  of  the  Connector/Arduino.

• Query  strings  (the  SQL  statements)  must  fit  into  memory.  This  is  because  the  class  uses  an  internal  buffer  for  building  data  packets  to  send  to  the  server.  It  is  suggested  long  strings  be  stored  in  program  memory  using  PROGMEM.

• Result  sets  are  read  one  row-­‐at-­‐a-­‐time  and  one  field-­‐at-­‐a-­‐time.

• The  combined  length  of  a  row  in  a  result  set  must  fit  into  memory.  The  connector  reads  one  packet-­‐at-­‐a-­‐time  and  since  the  Arduino  has  a  limited  data  size,  the  combined  length  of  all  fields  must  be  less  than  available  memory.

• Server  error  responses  are  processed  immediately  with  the  error  code  and  text  written  via

Serial.print.

**Changes from Previous Versions**

This  section  describes  the  changes  from  one  version  to  another  that  developers  will  need  to  know  in  order  to  convert  any  existing  code  to  use  the  new  version.  While  typically  no  major  changes  are  introduced  during  a  major.minor  version  release  cycle,  it  is  likely  changes  will  be  made  when  the  major  or  minor  version  is  incremented.

***Version  1.0.4-­‐>1.1.X***

The  theme  for  the  1.1.X  version  was  to  make  a  leap  forward  in  making  the  connector  easier  to  use  and  to  conform  to  newer  guidelines  for  writing  libraries  for  the  Arduino.  As  such,  many  of  the  method  names  changed  as  well  as  new  classes  were  added  to  help  improve  usability.  The  following  lists  the  major  changes  developers  need  to  know  in  order  to  adapt  the  new  version.

• ***New  Connection  Class***  :  A  new  MySQL\_Connection  class  was  added.  This  class  inherits  from  a  MySQL\_Packet  class,  which  contains  all  of  the  packet  handling  code.  Thus,  the  new  connection  class  is  smaller  with  only  a  few  methods  making  it  easier  to  use.  This  class  requires  an  instance  of  a  Client  class  compatible  with  the  Ethernet  or  WiFi  Arduino  libraries.  As  a  side  benefit,  the  connector  can  now  be  used  with  any  class  that  implements  the  same

methods  as  the  original  Ethernet  Client  method.  For  example,  if  you  bought  a  newer  Ethernet  shield  that  uses  a  new  chipset  (like  the  one  from  SeeedStudio),  you  can  use  it  with  the  connector  because  the  base  class  for  the  new  Ethernet2  library  is  the  same  as  the  Ethernet  Client  library.  Just  include  the  new  class  and  initialize  the  connector  with  a  new  instance  of  the  client.

• ***New  Cursor  Class***  :  A  new  MySQL\_Cursor  class  was  added.  This  class  permits  users  to

run  queries.  It  made  a  separate  class  mainly  to  remove  the  conditional  compilation  but  also

to  simplify  memory  handling.

• ***New  Examples***  :  The  original  code  examples  have  been  rewritten  to  correspond  with  the  new  documentation  examples.  There  are  also  a  host  of  new  examples  to  help  users  get  started  more  quickly.

• ***Simplified  Memory  Handling***:  The  original  code  required  the  caller  to  manage  memory  allocated  by  the  connector.  With  the  new  version,  users  need  not  include  the  free  memory  methods,  which  are  now  handled  internally  by  the  connector  and  cursor  class.

• ***Methods  Renamed***  :  In  order  to  conform  to  more  traditional  MySQL  connector  libraries,  several  methods  were  renamed.  The  following  summarizes  the  new  names.  Some  minor  functionality  is  slightly  different  as  shown.

|  |
| --- |
| Old  Method   Class   New  Method |
| mysql\_connect() MySQL\_Connection connect() |
| disconnect() MySQL\_Connection close() |
| is\_connected() MySQL\_Connection connected() |
| cmd\_query() MySQL\_Cursor execute()  ex: execute(query); |
| cmd\_query\_P() MySQL\_Cursor execute()  ex: execute(query, true); |
| free\_\*() MySQL\_Cursor close() |

**For More Information**

There  is  a  forum  setup  to  answer  questions  about  the  connector,  which  includes  questions  about  use

and  problems  using  it  [(http://forums.mysql.com/list.php?17](http://forums.mysql.com/list.php)5).

You  may  also  respond  to  my  blogs  [(http://drcharlesbell.blogspot.com](http://drcharlesbell.blogspot.com/)/),  but  keep  in

mind  some  of  these  entries  are  getting  quite  long  and  many  repeat  the  same  questions  over  and  over.

So  before  asking  your  question,  be  sure  you’ve  read  this  document  in  its  entirety  (especially  the  FAQ)

before  submitting  a  new  question.  Chances  are,  others  have  seen  your  problem  and  a  solution

already  exists.

I  will  accept  special  requests  emailed  to  me  directly  at [drcharlesbell@gmail.co](mailto:drcharlesbell@gmail.com)m  or  [chuck.bell@oracle.com,  but  I  reserve  the  right  to  delay  my  response  until  time  permits.  Thus,  do  not  expect  an  immediate  answer  (but  sometimes  I  will  respond  within  24  hours).](mailto:bell@oracle.com)