TINE Release 4.0 News

(May 6, 2011: That was the month that was !)

“What a long, strange trip it’s been ….”

Release 4.2.3

- Improvements in version 4.2.3
  - *Performance Adjustment* routines
  - *Exotic* concurrency problem and race condition fixed !
- MatLab API improvements !
- .NET news

Release 4.2.3

- Bug Fixes (C-Lib)
  - Client Side Links using CA_NETWORK **AND** CA_SYNCNOTIFY
    - Multi-threaded builds could land in a ‘*race-condition*’ causing a ‘double callback’ (thank you, Stefan!)
    - Initial bug fix (May 23) effectively considered CA_NETWORK **OR** CA_SYNCNOTIFY, causing ‘missed updates’ (thank you, Kai Brede!)
  - Server Side *deadlock* if a TCP client closes ungracefully
    - i.e. a TCP connection calling CloseLink() and ‘halting’ without any ‘*cycle activity*’.

Release 4.2.3

- Bug Fixes (java)
  - A notification problem with ‘DATACHANGE’ Links following a server restart was fixed (thank you, Elke).
  - A ‘race-condition’ which develops with TLink.execute() in a tight loop was fixed (thank you, David).
  - A data update problem (server side) for large payloads when the first MTU’s worth of data does not change was fixed. (thank you, Juergen).

Release 4.2.3

- Embellishments (C-Lib)
  - Routines Get/SetTransportRetryLimit() now available
    - Default = 2 is too large in some (*extreme*) scenarios (especially with TCP connections).
  - Routine SetSystemCleanupFunction() now available
    - Prototype: (void)(*fcn)(void);
    - Called as a last step during a ‘quit’ or ‘exit’.
    - Note: pure client applications had no chance to register a cleanup routine!

Release 4.2.3

- Embellishments (java)
  - TLink.execute() methods with ‘retryOnTimeout’ parameter now use a ‘hard’ timeout when

    retryOnTimeout = false.

    - Call returns ‘TErrorList.connection_timeout’ when given timeout interval expires (no ‘grace period’, no hidden retries).

Release 4.2.3

Example:

```
TLink sin2 = new TLink("/TEST/WinSineServer/SineGen0","Sine",sine_dout,null,TAccess.CA_READ);
for (int i=0; i<10; i++)
{
  t1 = System.currentTimeMillis();
  cc = sin2.execute(100,false);
  t2 = System.currentTimeMillis();
  System.out.println("call "+i+ " status : "+TErrorList.getErrorString(cc)+" in "+(t2-t1)+" msec");
}
```

```
call 0 status : connection timeout in 110 msec
call 1 status : connection timeout in 94 msec
call 2 status : connection timeout in 109 msec
call 3 status : connection timeout in 94 msec
call 4 status : connection timeout in 109 msec
call 5 status : connection timeout in 94 msec
call 6 status : connection timeout in 109 msec
call 7 status : connection timeout in 94 msec
call 8 status : connection timeout in 94 msec
call 9 status : connection timeout in 109 msec
```

Some clock Tick granularity, but very close to ‘100 msec’ !

From Last Time …

MatLab News

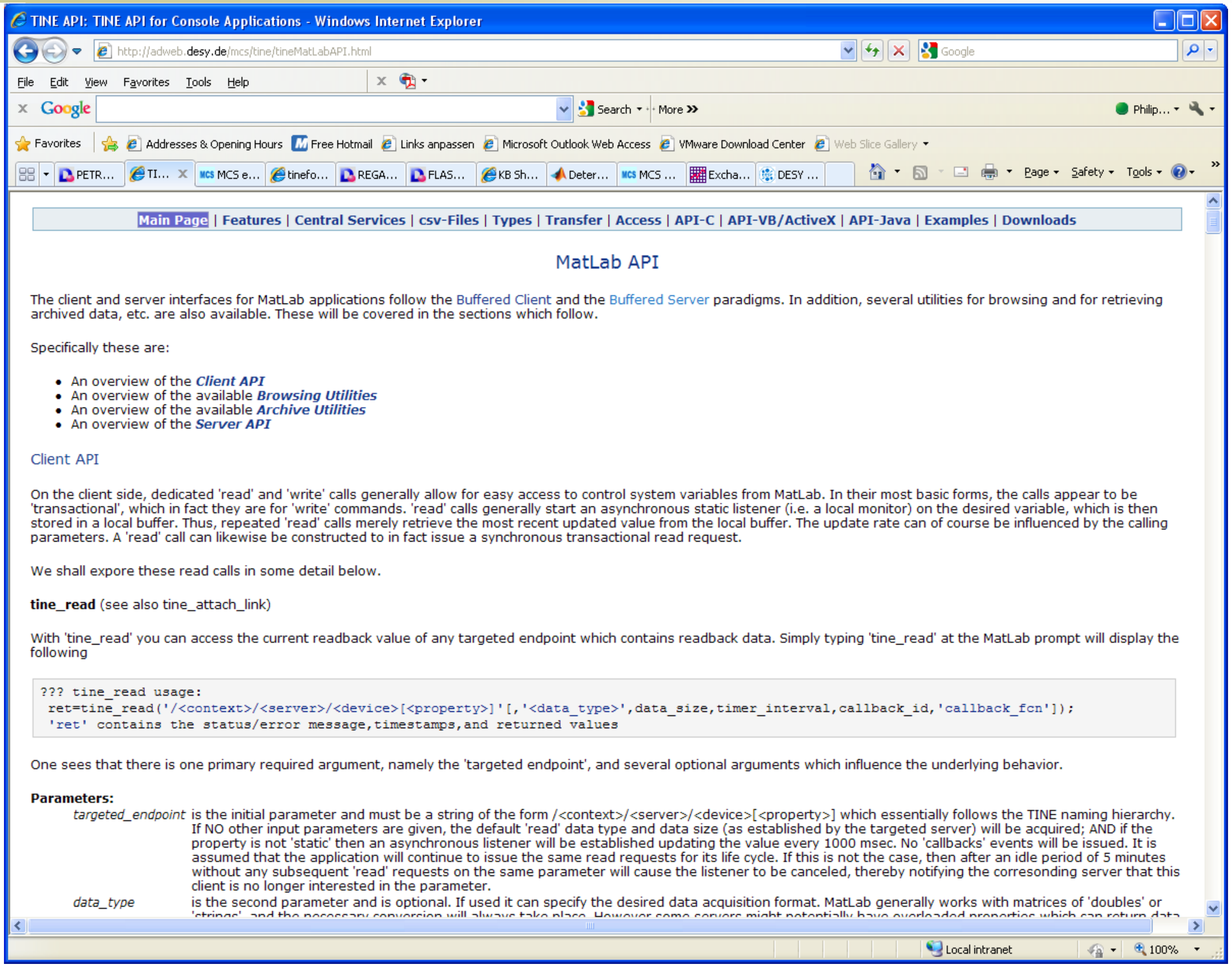
- **New routines:**
  - tine_eventdata(), tine_eventlist(), tine_eventservers(), tine_eventtriggers(), tine_eventproperties(), tine_eventcomment()
  - tine_history()
  - tine_read(), tine_write(), tine_writeread(), tine_call()
  - tine_callback.m
  - tine_debug
  - *Will be documented soon!* ←

http://tine.desy.de -> MatLab API

  - *Legacy routines:* tineread(), tinewrite(), tinewriteread() still work fine.

Documentation galore !

MatLab News …


TINE API: TINE API for Console Applications - Windows Internet Explorer
http://adweb.desy.de/mcs/tine/tineMatLabAPI.html
File Edit View Favorites Tools Help
Main Page | Features | Central Services | csv-Files | Types | Transfer | Access | API-C | API-VB/ActiveX | API-Java | Examples | Downloads
Local intranet
100%

MatLab API

The client and server interfaces for MatLab applications follow the Buffered Client and the Buffered Server paradigms. In addition, several utilities for browsing and for retrieving archived data, etc. are also available. These will be covered in the sections which follow.

Specifically these are:

- An overview of the ***Client API***
- An overview of the available ***Browsing Utilities***
- An overview of the available ***Archive Utilities***
- An overview of the ***Server API***

Client API

On the client side, dedicated 'read' and 'write' calls generally allow for easy access to control system variables from MatLab. In their most basic forms, the calls appear to be 'transactional', which in fact they are for 'write' commands. 'read' calls generally start an asynchronous static listener (i.e. a local monitor) on the desired variable, which is then stored in a local buffer. Thus, repeated 'read' calls merely retrieve the most recent updated value from the local buffer. The update rate can of course be influenced by the calling parameters. A 'read' call can likewise be constructed to in fact issue a synchronous transactional read request.

We shall expore these read calls in some detail below.

tine_read (see also tine_attach_link)

With 'tine_read' you can access the current readback value of any targeted endpoint which contains readback data. Simply typing 'tine_read' at the MatLab prompt will display the following

```
??? tine_read usage:
 ret=tine_read('/<context>/<server>/<device>[<property>]'[,'<data_type>',data_size,timer_interval,callback_id,'callback_fcn']);
 'ret' contains the status/error message,timestamps,and returned values
```

One sees that there is one primary required argument, namely the 'targeted endpoint', and several optional arguments which influence the underlying behavior.

Parameters:

targeted_endpoint	is the initial parameter and must be a string of the form /<context>/<server>/<device>[<property>] which essentially follows the TINE naming hierarchy. If NO other input parameters are given, the default 'read' data type and data size (as established by the targeted server) will be acquired; AND if the property is not 'static' then an asynchronous listener will be established updating the value every 1000 msec. No 'callbacks' events will be issued. It is assumed that the application will continue to issue the same read requests for its life cycle. If this is not the case, then after an idle period of 5 minutes without any subsequent 'read' requests on the same parameter will cause the listener to be canceled, thereby notifying the corresonding server that this client is no longer interested in the parameter.
data_type	is the second parameter and is optional. If used it can specify the desired data acquisition format. MatLab generally works with matrices of 'doubles' or 'strings', and the necessary conversion will always take place. However some servers might potentially have overloaded properties which can return data

From Last Time …

MatLab News

- **Planned:**
  - Write clients in MatLab with link callbacks (instead of polling)
    - Note:
      - asynchronous listeners already reduce burden on the server!
      - but the MatLab client must still poll locally !
  - Write servers in MatLab with the 'buffered server' API (a la LabView)
    - tine_attachserver() OR tine_registerServer(), tine_registerProperty(), etc.
    - tine_pushdata()
    - tine_handleCommand()

Mission Accomplished !

MatLab News

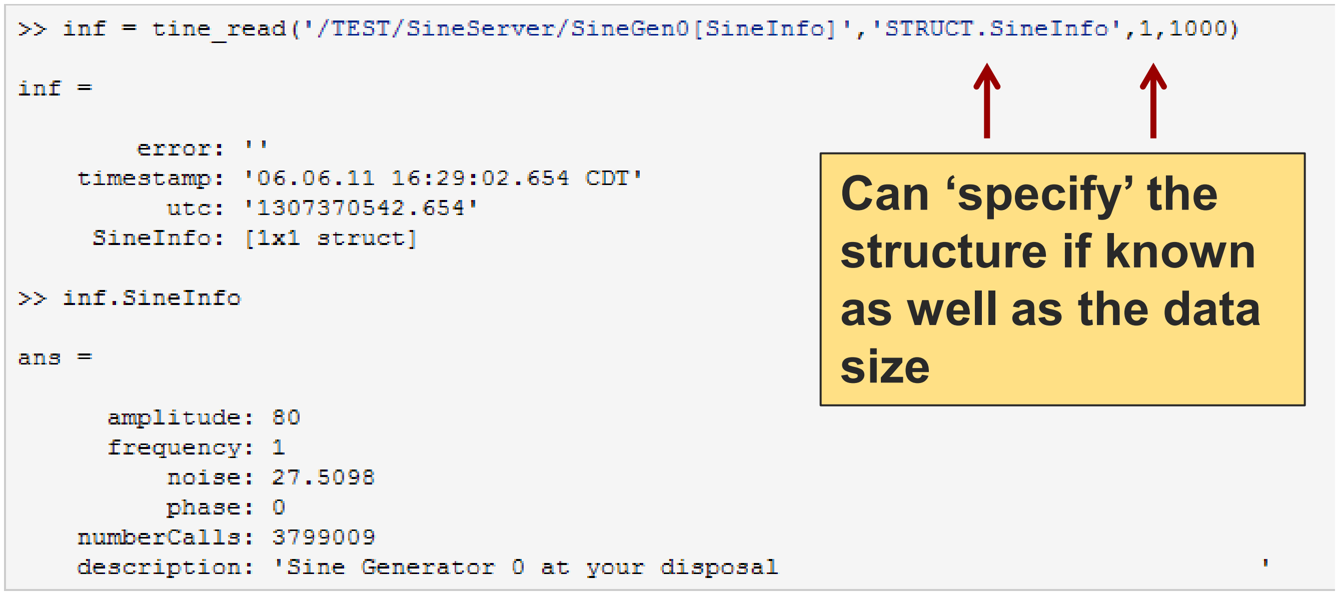
- Client Side
  - tine_read
  - tine_write
  - tine_writeread
  - tine_call
  - tine_attachlink ←
  - tine_closelink ←

Can make use of many more TINE Data types as well as Tagged Structures !

Takes callback ID and function as arguments !

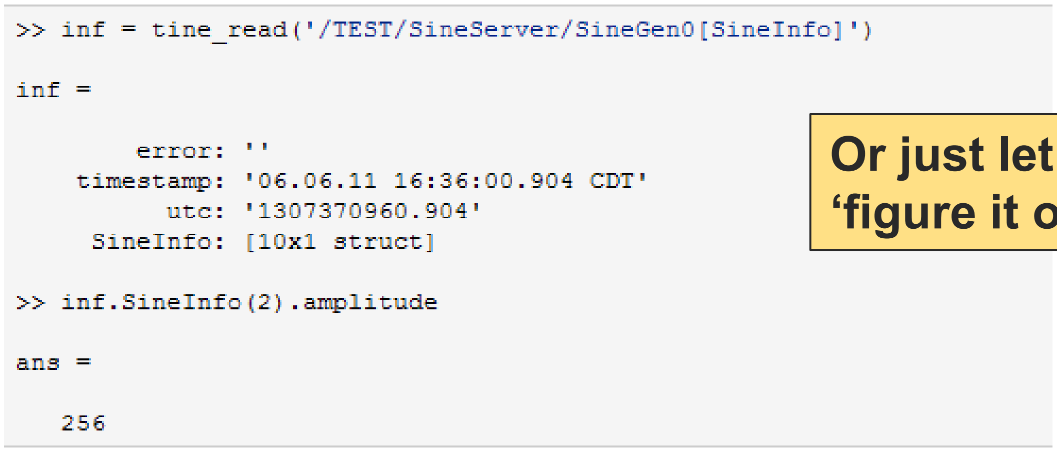
Closes a 'known' listening link !

MatLab News (reading structs)



```
>> inf = tine_read('/TEST/SineServer/SineGen0[SineInfo]','STRUCT.SineInfo',1,1000)

inf =

        error: ''
    timestamp: '06.06.11 16:29:02.654 CDT'
          utc: '1307370542.654'
     SineInfo: [1x1 struct]

>> inf.SineInfo

ans =

      amplitude: 80
      frequency: 1
          noise: 27.5098
          phase: 0
    numberCalls: 3799009
    description: 'Sine Generator 0 at your disposal                                    '
```

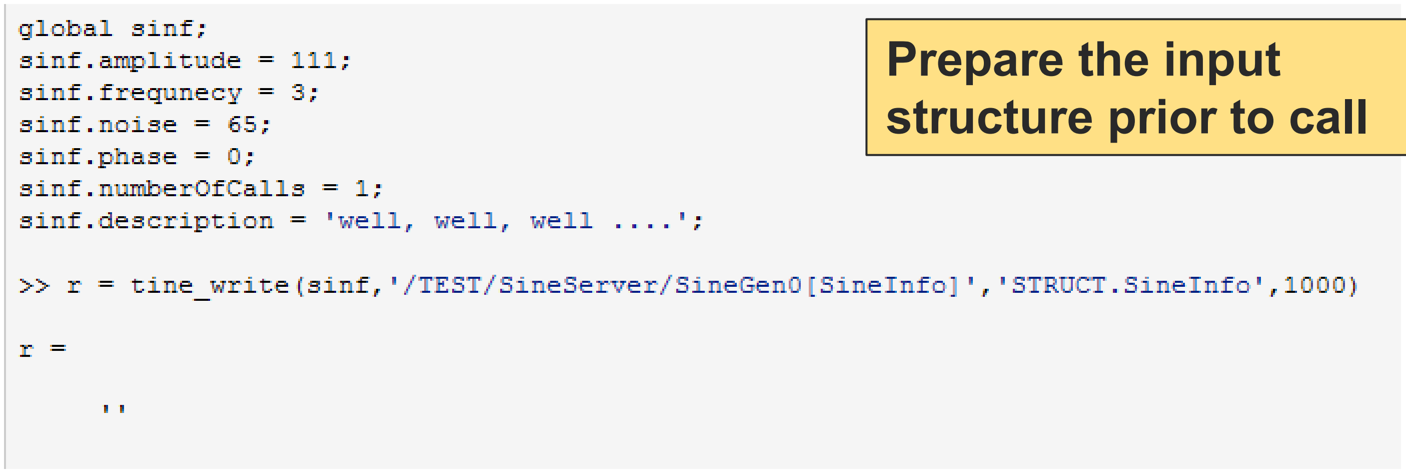
Can ‘specify’ the structure if known as well as the data size



```
>> inf = tine_read('/TEST/SineServer/SineGen0[SineInfo]')

inf =

        error: ''
    timestamp: '06.06.11 16:36:00.904 CDT'
          utc: '1307370960.904'
     SineInfo: [10x1 struct]

>> inf.SineInfo(2).amplitude

ans =

   256
```

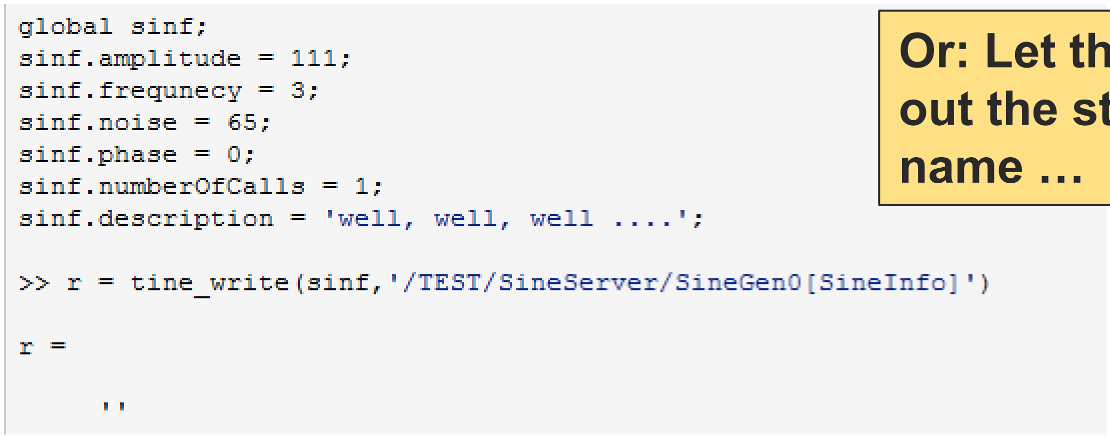
Or just let the call ‘figure it out’

MatLab News (writing structs)



```
global sinf;
sinf.amplitude = 111;
sinf.frequnecy = 3;
sinf.noise = 65;
sinf.phase = 0;
sinf.numberOfCalls = 1;
sinf.description = 'well, well, well ....';

>> r = tine_write(sinf,'/TEST/SineServer/SineGen0[SineInfo]','STRUCT.SineInfo',1000)

r =

     ''
```

Prepare the input structure prior to call



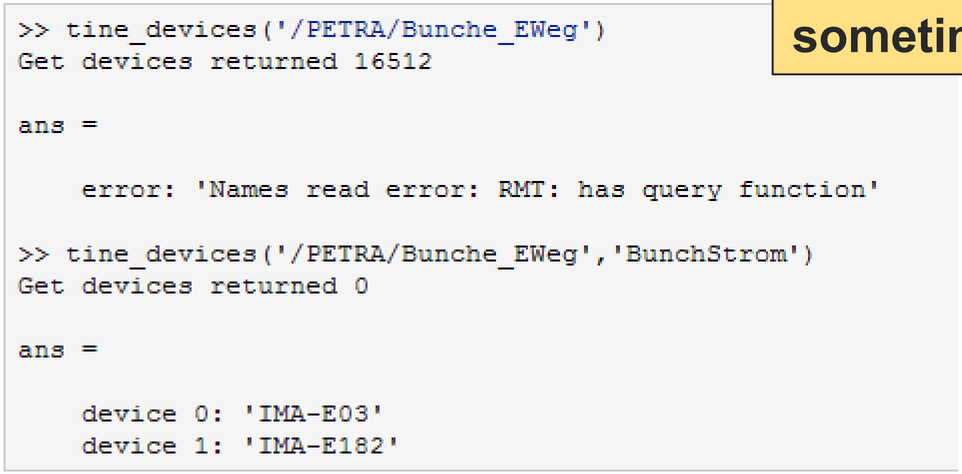
```
global sinf;
sinf.amplitude = 111;
sinf.frequnecy = 3;
sinf.noise = 65;
sinf.phase = 0;
sinf.numberOfCalls = 1;
sinf.description = 'well, well, well ....';

>> r = tine_write(sinf,'/TEST/SineServer/SineGen0[SineInfo]')

r =

     ''
```

Or: Let the call figure out the structure tag name …

MatLab News

- Browsing
  - tine_contexts
  - tine_servers
  - tine_devices
  - tine_properties

Example (tine_devices): some optional arguments are sometimes important!



```
>> tine_devices('/PETRA/Bunche_EWeg')
Get devices returned 16512

ans =

    error: 'Names read error: RMT: has query function'

>> tine_devices('/PETRA/Bunche_EWeg','BunchStrom')
Get devices returned 0

ans =

    device 0: 'IMA-E03'
    device 1: 'IMA-E182'
```

MatLab News

- Archive Calls
  - tine_history
  - tine_eventdata
  - tine_eventtriggers
  - tine_eventproperties
  - tine_eventservers
  - tine_eventlist
  - tine_eventcomment

MatLab News

- Server API
  - tine_attach_server
  - tine_push_data
  - tine_attach_handler
  - tine_register_fec
  - tine_register_server
  - tine_register_device
  - tine_register_property
  - tine_register_type (fuer TINE Structures)

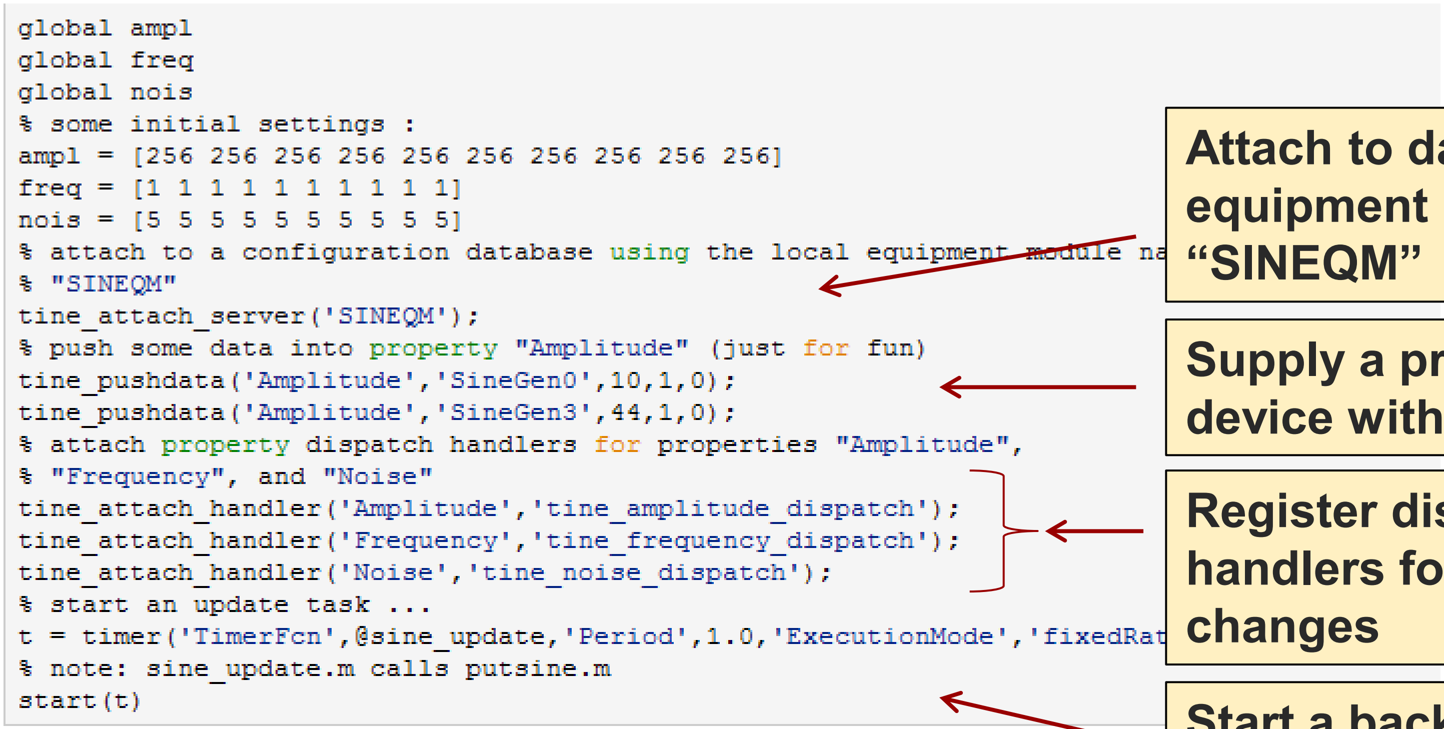
$\leftarrow$ **Initialization and Startup**

$\leftarrow$ **Supplying data and reacting to commands**

Or use configuration files

MatLab News

MatLab server using configuration files …



```
global ampl
global freq
global nois
% some initial settings :
ampl = [256 256 256 256 256 256 256 256 256 256]
freq = [1 1 1 1 1 1 1 1 1 1]
nois = [5 5 5 5 5 5 5 5 5 5]
% attach to a configuration database using the local equipment module na
% "SINEQM"
tine_attach_server('SINEQM');
% push some data into property "Amplitude" (just for fun)
tine_pushdata('Amplitude','SineGen0',10,1,0);
tine_pushdata('Amplitude','SineGen3',44,1,0);
% attach property dispatch handlers for properties "Amplitude",
% "Frequency", and "Noise"
tine_attach_handler('Amplitude','tine_amplitude_dispatch');
tine_attach_handler('Frequency','tine_frequency_dispatch');
tine_attach_handler('Noise','tine_noise_dispatch');
% start an update task ...
t = timer('TimerFcn',@sine_update,'Period',1.0,'ExecutionMode','fixedRat
% note: sine_update.m calls putsine.m
start(t)
```

Attach to database via equipment module “SINEQM”

Supply a property and device with data

Register dispatch handlers for settings changes

Start a background task

MatLab News

MatLab server without configuration files …

```
tine_register_fec('MLFEC','TEST','TEST','MatLab test fec','here','none','me',44);
tine_register_server('MLSineServer','MLEQM',10);
tine_register_property('Amplitude',1,'float',10,'float',512,1,'V','READ|WRITE','Sine Amplitude','CHANNEL');
tine_register_property('Frequency',1,'float',10,'float',50,1,'Hz','READ|WRITE','Sine Frequency','CHANNEL');
tine_register_property('Phase',1,'float',10,'float',6.28,0,' ','READ|WRITE','Sine Phase','CHANNEL');
tine_register_property('Noise',1,'float',10,'float',50,0,'V','READ|WRITE','Sine Noise','CHANNEL');
tine_register_property('Sine',0,'null',1024,'float',512,1,'V','READ','Sine Curve','SPECTRUM');
tine_register_device('SineGen0',0);
tine_register_device('SineGen1',1);
tine_register_device('SineGen2',2);
tine_register_device('SineGen3',3);
tine_register_device('SineGen4',4);
tine_register_device('SineGen5',5);
tine_register_device('SineGen6',6);
tine_register_device('SineGen7',7);
tine_register_device('SineGen8',8);
tine_register_device('SineGen9',9);
global ampl;
global freq;
global nois;
freq = [1 1 1 1 1 1 1 1 1 1];
ampl = [256 256 256 256 256 256 256 256 256 256];
nois = [5 5 5 5 5 5 5 5 5 5];
tine_pushdata('Amplitude','SineGen0',10,1,0);
tine_pushdata('Amplitude','SineGen2',44,1,0);
tine_attach_handler('Amplitude','tine_amplitude_dispatch');
tine_attach_handler('Frequency','tine_frequency_dispatch');
tine_attach_handler('Noise','tine_noise_dispatch');
t = timer('TimerFcn',@sine_update,'Period',1.0,'ExecutionMode','fixedRate');
start(t);
```

Supply all relevant information directly in MatLab code

MatLab News

MatLab Sine Server: updating the Sine curve :

```
function cc = putsine(DEV)
global ampl
global freq
global nois
% get the array index according to the device name
idx = get_sine_device_index(DEV);
r = 0:1:1024;
% use the correct amplitude, frequency, and noise array elements
% for the calcuation
v = ampl(idx) * sin(r *  2 * freq(idx) * pi / 1024);
v = v + (nois(idx) * randn(1,size(v,2)));
% push the results into the underlying property buffer
cc = tine_pushdata('Sine',DEV,v);
```

MatLab Sine Server: reacting to an amplitude setting change :

```
function ret = tine_amplitude_dispatch(PRP,DEV,DATA)
global ampl
% just get the new value and accept it
idx = get_sine_device_index(DEV)
ampl(idx) = DATA;
% push the new value into the corresponding read buffer:
ret = tine_pushdata(PRP,DEV,DATA,1,0);
```

MatLab News

MatLab Sine Server: registering a structure :

```
global inf;
inf.amplitude = 100;
inf.frequency = 1;
inf.noise = 50;
inf.phase = 0;
inf.description = 'just another sine curve';
tine_register_type('MlabInf',inf);
```

Will use the structure tag ‘MlabInf’ in this example

Now register a property to use this new ‘type’:

```
tine_register_property('SineInfo',1,'struct.MlabInf',1,'struct.MlabInf',0,0,'none','READ|WRITE','Sine info','SPECTRUM');
```

‘push’ data when you need to :

```
tine_pushdata('SineInfo','SineGen0',inf);
```

A property dispatch handler will also see an incoming structure (for atomic changes) !

MatLab News

- MatLab ‘mex’ routines tested on
  - Win32
  - Win64
    - (but the Terminal Servers seem to have a firewall issue with servers)
  - Linux32
  - Linux64

.NET News

- Bug fix in TDataType.putData() when passing a scalar by value
- New server registration routines
- More integrated documentation
- API coming soon to the Web Site

doocs-tine issues

- jddd and ddd like to rely on *device name* and *property* to simply return relevant data
  - Good idea for panel/widget programing
  - e.g.
    - a rich client would say give me *1 float value* for /PETRA/BPM/BPM_SWR_13[Orbit.X]
    - a panel client would say give me *the data* for /PETRA/BPM/BPM_SWR_13[Orbit.X]
  - But
    - A tine contract will always specify a data type and a data size!
    - Solutions:
      - 1) 1st query the property to see what it delivers, then do that!
      - 2) specify a **buffer capacity** (in bytes) and data type **CF_DEFAULT**
    - Solution 2) involves no extra traffic and is preferred.
    - Problem: if buffer capacity is not sufficient the call receives 'buffer_too_small'
      - C-Lib handles this (default capacity = 64 bytes) !
      - Java does not (as yet)! (default capacity was 64 Kbytes) !

doocs-tine issues

Reacting to ‘buffer_too_small’ in java:

```
TDataType d = new TDataType(64,TFormat.CF_DEFAULT);
TLink lnk1 = new TLink("/TEST/SineServer/SineGen0","Sine",
                       d,null,TAccess.CA_READ);
lnk1.attach(TMode.CM_TIMER,instance,1000);
```

Much too small for a sine curve (8 K floats)

Possible callback strategy :

```
public void callback(TLink link)
{
  if (link.getLinkStatus() == TErrorList.buffer_too_small)
  { // default data set larger than my suggestion of 64 bytes !
    TPropertyQuery[] tp = TQuery.getPropertyInformation(link.getContext(), link.getDeviceServer(),
        link.getDeviceName(), link.getProperty(), 500);
    if (tp != null)
    { // this should always work as we have just heard from a running server !
      link.setOutputDataObject(new TDataType(tp[0].prpSize,tp[0].prpFormat));
      link.attach(TMode.CM_TIMER,instance,1000);
    }
    return;
  }
  TDataType tdt = link.getOutputDataObject();
  switch (tdt.getFormat())
  {
    case TFormat.CF_SPECTRUM:
```

Will re-attach with the ‘correct’ parameters !

doocs-tine issues

- Note: *tine* needs to allocate a data buffer (likely 2 buffers) on both the client and server side to manage contracts and connections.
  - necessary for persistent links
    - Common links to same contract, etc.
    - Double buffering for ‘DATACHANGE’, etc.
- Different from a SunRPC transaction which goes out of memory when it completes.

doocs-tine issues

- Potential points of confusion and inefficiencies when using CF_DEFAULT:
  - A *tine* property can be overloaded !
    - e.g. deliver a timestamp as UTC ***long int*** or as a ***string***
    - e.g. deliver different ***structures***
    - => CF_DEFAULT gives only the '*preferred*' data type and size.
  - the default size could be much larger than necessary!
    - wasteful of main memory for a monitored link !
- These points are currently being addressed!

doocs-tine issues

- Security
  - tine uses ‘user name’ and/or network address
    - at the server level
    - at the property level
    - at the device level
    - (Can also use ‘access locks’ -> application level).
  - doocs uses gid + uid
    - at the server level
    - at the property level
    - at the device level
    - pid ?

doocs-tine issues

- Security
  - tine-to-doocs via tine
    - tine security turned off
    - tries to map user name into gid/uid
  - Problems:
    - A (middle-layer) FEC always uses its FEC name as the user name (regardless of the logged in user).
      - Note: this strategy sometimes has tremendous advantages!
    - Workaround:
      - SetUser("DOOCSADM");
      - Make the call
      - SetUser(FEC NAME);