# TINE Release 4.0 News

(Dec 17, 2010: That was the month that was !)

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

# Release 4.2.0

- C (Standard) Lib is now at 4.2.0; Java close behind
  - Some additional ‘Save-and-Restore’ routines
  - Callback group synchronization

- Important Bug Fixes / Improvements (C-Lib):
  - Problem initializing cycle trigger synchronization fixed
  - Environment variable FEC_LOG_COMMANDS available to override the default setting (TRUE).
    - API alternative: SetPutCommandsInFeclog()
  - Wait for 1st notification on dependent sync links.
    - An ‘ExecLink’ immediately following an ‘AttachLink’ with the same link parameters
  - Command Line Routines: tsend, tputget, tsendrecv
    - Fixed parsing problem with a single *negative* input value

# Release 4.2.0

- **Important Bug Fixes / Improvements (Java):**
  - Legacy bug (client side) where ‘polling Interval’ was interpreted as a signed short!
    - E.g. a value of 60000 msec (1 minute) became -27233 which was less than the default allowed 10 msec -> converted to 10 msec!
  - Default minimum polling interval now 100 msec.
  - Call setHistoryCycleInterval() automatically if history registration provides a fast polling interval
  - Move the Time Synchronization code to the TDataTime Class.
    - Servers call TDataTime.systemStartGlobalSynchronization() automatically.
    - Pure Clients can now call TDataTime.systemStartGlobalSynchronization().

# Release 4.2.0

## Save and Restore Routines

```
int RestorePropertyValues ( const char *  eqmName,
                            const char *  prpName,
                            void *        values,
                            short         format,
                            int           size
                          )
```

Retrieves the value settings of the property name given from disk.

Using this routine will restore the given values of the named property from the disk file named <prpName>-settings.csv.

**Parameters:**

| | |
|---|---|
| *eqmName* | is the local equipment module name. |
| *prpName* | is the name of the property whose values are to be restored |
| *values* | is a reference to the values to be restored |
| *format* | is the format of the values to be restored (simple format type) |
| *size* | is the array size (not the size in bytes) of the values to be restored. |

**Returns:**

0 if successful, otherwise a TINE completion code

```
int SavePropertyValues ( const char *  eqmName,
                         const char *  prpName,
                         void *        values,
                         short         format,
                         int           size
                       )
```

Saves value settings of the property name given onto disk.

Using this routine will back out the given values of the named property to a disk file named <prpName>-settings.csv.

**Parameters:**

| | |
|---|---|
| *eqmName* | is the local equipment module name. |
| *prpName* | is the name of the property whose values are to be stored |
| *values* | is a reference to the values to be stored |
| *format* | is the format of the values to be stored (simple format type) |
| *size* | is the array size (not the size in bytes) of the values to be stored. |

**Returns:**

0 if successful, otherwise a TINE completion code

# Release 4.2.0

## Range Checking Routines

```
int AssertRangeValid ( const char *  eqmName,
                       const char *  prpName,
                       DTYPE *       din,
                       int           enforceLimits
                     )
```

Helper routine to check input data against registered range limits.

This routine returns TRUE if the given input does not violate the registered range settings for the property given. Any errors in input will result in a 'TRUE' being returned. If the 'enforceLimits' parameter is 'TRUE' then the routine will always return TRUE but will mutate the **DTYPE** object so that any range exceptions are set to the registered maximum or minimum values.

This helper routine will only consider input data objects supplying a single valued 'primitive' numerical format type.

**Parameters:**

| | |
|---|---|
| *eqmName* | is the local equipment module name. |
| *prpName* | is the name of the property whose values are to be restored |
| *din* | is a reference to input data to be checked against the registered range settings. |
| *enforceLimits* | will insert the corresponding maximum or minimum value into the din object reference should any range exception be detected. |

**Returns:**

TRUE if no range violation is detected.

References **DUNION::bptr**, **DTYPE::dArrayLength**, **DTYPE::data**, **DTYPE::dFormat**, **DUNION::dptr**, **DUNION::fptr**, **GetRegisteredPropertyListStruct( )**, **DUNION::llptr**, DUNION::lptr, and **DUNION::sptr**.

# Release 4.2.0

## Save and Restore Routines: Saving

```
    return 0;
case PRP_FREQUENCY:
    if (din->dArrayLength > 0)
    { /* input data => require write access */
      if (!(access & CA_WRITE)) return illegal_read_write;
      if ((cc=getValuesAsFloat(din,&fval,1)) != 0) return cc;
      if (fval < 1 || fval > 100) return out_of_range;
      sineInfoTable[devnr].frequency = fval;
      for (i=0; i<NUM_DEVICES; i++)
        mcarray[i] = sineInfoTable[i].frequency;
      SavePropertyValues("SINEQM","Frequency",mcarray,CF_FLOAT,NUM_DEVICES);
    }
    if (dout->dArrayLength > 0)
    { /* prepare multichannel array */
      for (i=0; i<NUM_DEVICES; i++)
        mcarray[i] = sineInfoTable[i].frequency;
      if ((cc=putValuesFromFloatEx(dout,mcarray,NUM_DEVICES,devnr)) != 0) return cc;
    }
    dout->dTimeStamp = dts;
    return 0;
case PRP_PHASE:
    if (din->dArrayLength > 0)
    { /* input data => require write access */
      if (!(access & CA_WRITE)) return illegal_read_write;
      if ((cc=getValuesAsFloat(din,&fval,1)) != 0) return cc;
      if (!AssertRangeValid("SINEQM",devProperty,din,TRUE)) return out_of_range;
      sineInfoTable[devnr].phase = fval;
      for (i=0; i<NUM_DEVICES; i++)
        mcarray[i] = sineInfoTable[i].phase;
      SavePropertyValues("SINEQM","Phase",mcarray,CF_FLOAT,NUM_DEVICES);
    }
    if (dout->dArrayLength > 0)
    { /* prepare multichannel array */
      for (i=0; i<NUM_DEVICES; i++)
        mcarray[i] = sineInfoTable[i].phase;
      if ((cc=putValuesFromFloatEx(dout,mcarray,NUM_DEVICES,devnr)) != 0) return cc;
    }
    return 0;
case PRP_NOISE:
    if (din->dArrayLength > 0)
    { /* input data => require write access */
      if (!(access & CA_WRITE)) return illegal_read_write;
      if ((cc=getValuesAsFloat(din,&fval,1)) != 0) return cc;
      if (fval < 0 || fval > 100) return out_of_range;
```

# Release 4.2.0

**Save and Restore Routines: restoring …**

- At initialization time:

```
if (RestorePropertyValues("SINEQM","Amplitude",fv,CF_FLOAT,NUM_DEVICES) == 0)
{
  for (i=0; i<NUM_DEVICES; i++) sineInfoTable[i].amplitude = fv[i];
}
if (RestorePropertyValues("SINEQM","Frequency",fv,CF_FLOAT,NUM_DEVICES) == 0)
{
  for (i=0; i<NUM_DEVICES; i++) sineInfoTable[i].frequency = fv[i];
}
if (RestorePropertyValues("SINEQM","Phase",fv,CF_FLOAT,NUM_DEVICES) == 0)
{
```

# Release 4.2.0

- What can be done automatically?
  - Property registered as ‘SaveAndRestore’
  - At initialization:
    - Reads ‘saved’ values at startup
    - Call eqm handler with saved values
  - If Command:
    - Call ‘Save’ if contract successful

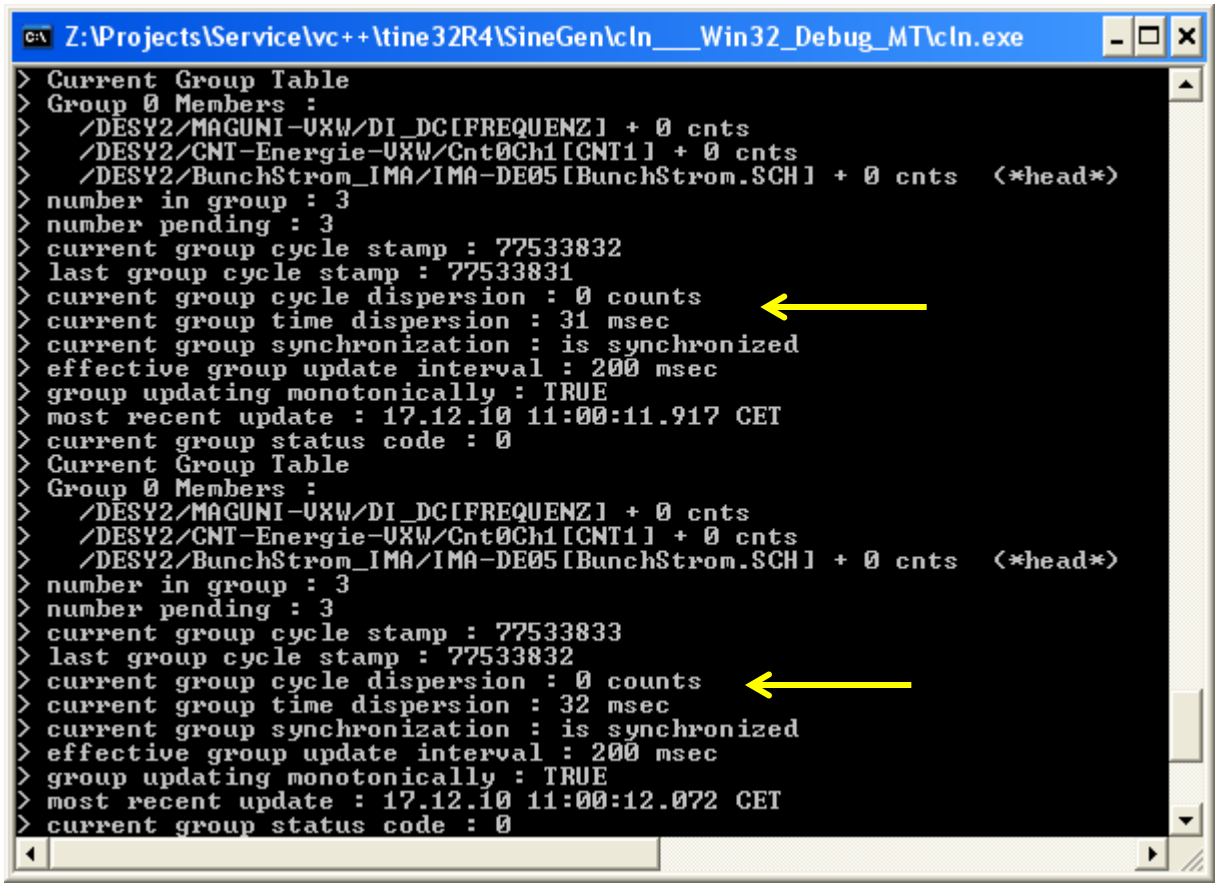
# Release 4.2.0

- Callback Synchronization
  - Currently:
    - Group notification achieved via CM_GROUPED bit in access mode
    - Wait for all group members prior to callback notification
  - In addition:
    - CM_SYNCGROUP bit
      - Minimize time ‘dispersion’ and cycle count ‘dispersion’
    - Retrieve cycle ‘offset’ and other group info
      - GetCallbackGroup()

# Release 4.2.0

```
int d2BunCurId = -1;
int d2EngCntrId = -1;
int d2VacAvePId = -1;
int d2OrbId = -1;
int d2FreqId = -1;
float d2bunCur = 0;
int d2engCntr = 0;
int d2Freq = 0;
float d2VacAveP = 0;
float d2orb[24];

void d2grpCb(int id,int cc)
{
  GrpTblEntry *g=GetCallbackGroup(d2grpCb);
  static int cnt=0;
  if ((++cnt % 10) == 0 || g->grpBndWdthC == 0)
  {
    outputConnectionGroups();
  }
  return;
}
int d2grp_test()
{
  DTYPE dout;
  int i, id, cc;
  int mode = CM_TIMER|CM_GROUPED|CM_SYNCGROUP;

  dout.dFormat = CF_FLOAT;
  dout.dArrayLength = 1;
  dout.data.fptr = &d2bunCur;
  dout.dTag[0] = 0;

  d2bunCurId = AttachLink("/DESY2/BunchStrom_IMA/IMA-DE05","BunchStrom.SCH",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  dout.dFormat = CF_INT32;
  dout.dArrayLength = 1;
  dout.data.lptr = &d2engCntr;

  d2EngCntrId = AttachLink("/DESY2/CNT-Energie-VXW/Cnt0Ch1","CNT1",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  dout.dFormat = CF_INT32;
  dout.dArrayLength = 1;
  dout.data.lptr = &d2Freq;

  d2FreqId = AttachLink("/DESY2/MAGUNI-VXW/DI_DC","FREQUENZ",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  return 0;
}
```

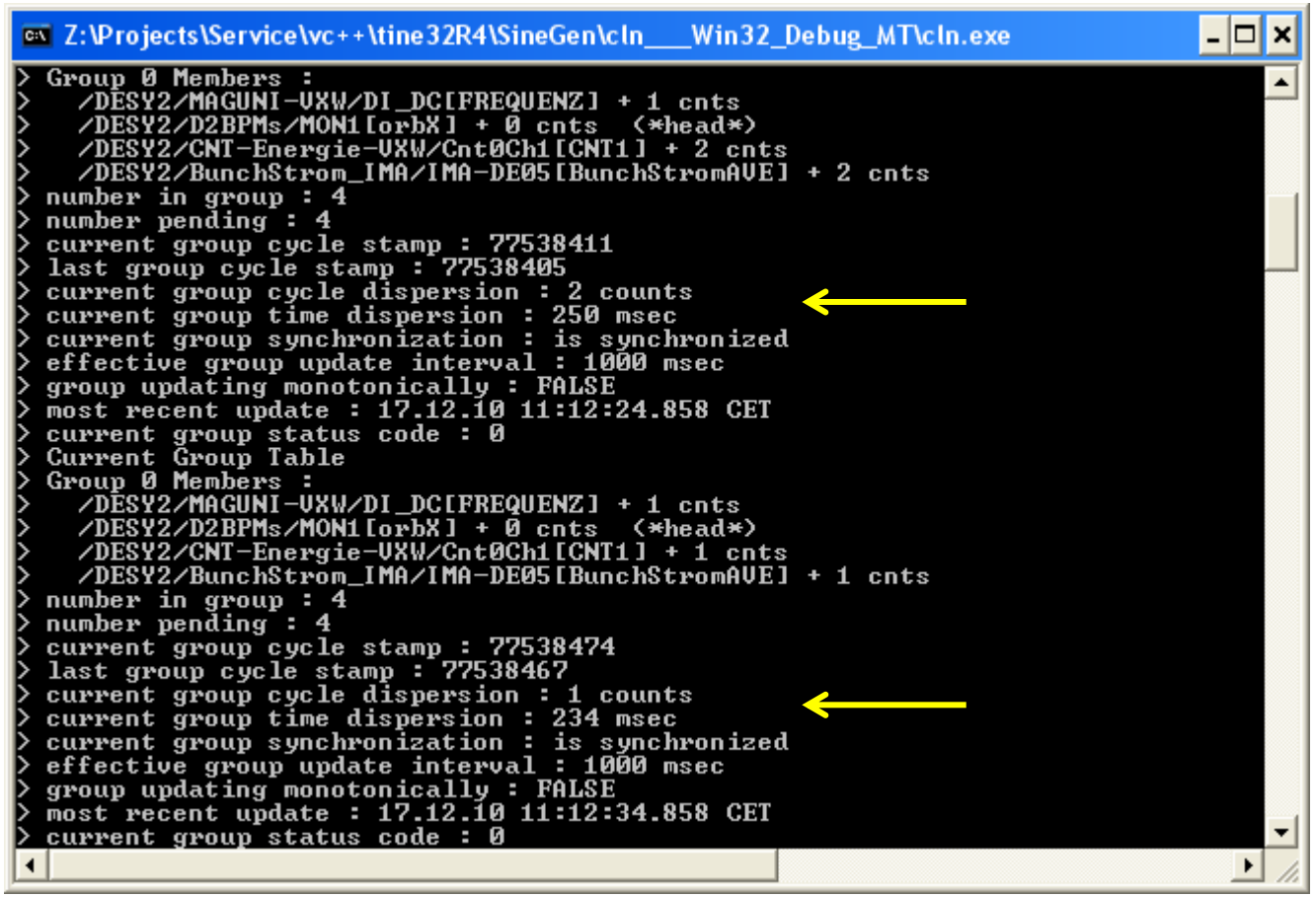
# Release 4.2.0


Z:\Projects\Service\vc++\tine32R4\SineGen\cln___Win32_Debug_MT\cln.exe
> Current Group Table
> Group 0 Members :
>   /DESY2/MAGUNI-VXW/DI_DC[FREQUENZ] + 0 cnts
>   /DESY2/CNT-Energie-VXW/Cnt0Ch1[CNT1] + 0 cnts
>   /DESY2/BunchStrom_IMA/IMA-DE05[BunchStrom.SCH] + 0 cnts  (*head*)
> number in group : 3
> number pending : 3
> current group cycle stamp : 77533832
> last group cycle stamp : 77533831
> current group cycle dispersion : 0 counts
> current group time dispersion : 31 msec
> current group synchronization : is synchronized
> effective group update interval : 200 msec
> group updating monotonically : TRUE
> most recent update : 17.12.10 11:00:11.917 CET
> current group status code : 0
> Current Group Table
> Group 0 Members :
>   /DESY2/MAGUNI-VXW/DI_DC[FREQUENZ] + 0 cnts
>   /DESY2/CNT-Energie-VXW/Cnt0Ch1[CNT1] + 0 cnts
>   /DESY2/BunchStrom_IMA/IMA-DE05[BunchStrom.SCH] + 0 cnts  (*head*)
> number in group : 3
> number pending : 3
> current group cycle stamp : 77533833
> last group cycle stamp : 77533832
> current group cycle dispersion : 0 counts
> current group time dispersion : 32 msec
> current group synchronization : is synchronized
> effective group update interval : 200 msec
> group updating monotonically : TRUE
> most recent update : 17.12.10 11:00:12.072 CET
> current group status code : 0

# Release 4.2.0

```
float d2bunCur = 0;
int d2engCntr = 0;
int d2Freq = 0;
float d2VacAveP = 0;
float d2orb[24];

void d2grpCb(int id,int cc)
{
  GrpTblEntry *g=GetCallbackGroup(d2grpCb);
  static int cnt=0;
  if ((++cnt % 10) == 0 || g->grpBndWdthC == 0)
  {
    outputConnectionGroups();
  }
  return;
}
int d2grp_test()
{
  DTYPE dout;
  int i, id, cc;
  int mode = CM_TIMER|CM_GROUPED|CM_SYNCGROUP;

  dout.dFormat = CF_FLOAT;
  dout.dArrayLength = 1;
  dout.data.fptr = &d2bunCur;
  dout.dTag[0] = 0;

  d2bunCurId = AttachLink("/DESY2/BunchStrom_IMA/IMA-DE05","BunchStromAVE",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  dout.dFormat = CF_INT32;
  dout.dArrayLength = 1;
  dout.data.lptr = &d2engCntr;

  d2EngCntrId = AttachLink("/DESY2/CNT-Energie-VXW/Cnt0Ch1","CNT1",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  dout.dFormat = CF_INT32;
  dout.dArrayLength = 1;
  dout.data.lptr = &d2Freq;

  d2FreqId = AttachLink("/DESY2/MAGUNI-VXW/DI_DC","FREQUENZ",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  dout.dFormat = CF_FLOAT;
  dout.dArrayLength = 24;
  dout.data.fptr = d2orb;

  d2OrbId = AttachLink("/DESY2/D2BPMs/MON1","orbX",&dout,NULL,CA_READ,1000,d2grpCb,mode);

  return 0;
}
```

# Release 4.2.0


Z:\Projects\Service\vc++\tine32R4\SineGen\cln___Win32_Debug_MT\cln.exe
> Group 0 Members :
>   /DESY2/MAGUNI-VXW/DI_DC[FREQUENZ] + 1 cnts
>   /DESY2/D2BPMs/MON1[orbX] + 0 cnts  (*head*)
>   /DESY2/CNT-Energie-VXW/Cnt0Ch1[CNT1] + 2 cnts
>   /DESY2/BunchStrom_IMA/IMA-DE05[BunchStromAVE] + 2 cnts
> number in group : 4
> number pending : 4
> current group cycle stamp : 77538411
> last group cycle stamp : 77538405
> current group cycle dispersion : 2 counts
> current group time dispersion : 250 msec
> current group synchronization : is synchronized
> effective group update interval : 1000 msec
> group updating monotonically : FALSE
> most recent update : 17.12.10 11:12:24.858 CET
> current group status code : 0
> Current Group Table
> Group 0 Members :
>   /DESY2/MAGUNI-VXW/DI_DC[FREQUENZ] + 1 cnts
>   /DESY2/D2BPMs/MON1[orbX] + 0 cnts  (*head*)
>   /DESY2/CNT-Energie-VXW/Cnt0Ch1[CNT1] + 1 cnts
>   /DESY2/BunchStrom_IMA/IMA-DE05[BunchStromAVE] + 1 cnts
> number in group : 4
> number pending : 4
> current group cycle stamp : 77538474
> last group cycle stamp : 77538467
> current group cycle dispersion : 1 counts
> current group time dispersion : 234 msec
> current group synchronization : is synchronized
> effective group update interval : 1000 msec
> group updating monotonically : FALSE
> most recent update : 17.12.10 11:12:34.858 CET
> current group status code : 0

# Release 4.2.0

## New Server Side Synchronization Routines:

**void SetSystemStampDelay ( int cycleDelay )**

Establishes the system cycle delay.

If a server's context has a registered 'CYCLER' then all read data will be tagged with the incoming system cycle number. If it is known a priori that due to hard i/o latency the application of the cycle tag needs to be delayed by some value, then this routine may be used to establish such a cycle delay value (in milliseconds).

**Parameters:**

*cycleDelay* is the desired cycle delay (milliseconds), which will must elapse before the incoming cycle number from the registered CYCLER is to be applied to all readback data. (default = 0).

**See also:**

GetSystemStampDelay

**void SetSystemStampOffset ( int cycleOffset )**

Establishes a system cycle offset.

If a server's context has a registered 'CYCLER' then all read data will be tagged with the incoming system cycle number. If it is known a priori that due to hard i/o latency the cycle tag needs to be offset by some value, then this routine may be used to establish such an offset.

**Parameters:**

*cycleOffset* is the desired cycle offset (counts) to be applied to the incoming cycle number from the registered CYCLER. (Default = 0).

**See also:**

GetSystemStampOffset

# Release 4.2.0

## Old (forgotten?) Server Side Synchronization Routines:

```
int RegisterCycleTriggerFunction ( CYCBFCNP fcn,
                                   char *   eqm,
                                   char *   prpLst,
                                   void *   reference
                                 )
```

Registers a cycle trigger callback dispatch function.

If a CYCLER is running in a server's context, then the server will receive 'Cycle Number' events scheduled by the designated CYCLER server. The cycle number will make use of the 'System Data Stamp' to tag all data sets obtained from the server. A server can also register a trigger function dispatch routine (or routines) to be called when a 'Cycle Number' event occurs. The dispatch routines will be called prior to setting the 'System Stamp' to the new Cycle Number, which will be set following the execution of all dispatch routines. Optionally, the server can provide a property (or list of properties) to be scheduled following the dispatch execution. This will ensure that such properties will be called immediately following dispatch execution AND contain the most recent Cycle Number as the 'System Data Stamp'.

**Parameters:**

| | |
|---|---|
| *fcn* | is a reference to the cycle trigger dispatch routine to be called following the reception of a new Cycle Number. This must have the prototype: void (*fcn)(int cycleNumber,int cycleStatus,void *reference). Thus, the dispatch routine will be called with the current cycle number, the cycle status (possibly 'link_timeout' if no cycle is received within the assigned globals heartbeat (see SetGlobalsHeartbeat)), and an optional 'reference', which is a void pointer of the caller's choosing (and will be returned to the caller in the dispatch routine). |
| *eqm* | the local equipment module name of the desired central server (e.g. "BPMEQM") |
| *prpLst* | is the property or properties which are to be 'scheduled' following the execution of the dispatch routine. If more than a single property is to be scheduled, this should be a string containing a comma separated list. |
| *reference* | is a caller supplied void pointer which will be returned to the called in the dispatch routine. |

**Returns:**

0 if successful, otherwise a TINE completion code

**Example:**

```
#define EQMTAG "TSTEQM"
#define PRP_CYCLE      1
int cycleNumber = 0;
int tsteqm(char *devName,char *devProperty,DTYPE *dout, DTYPE *din,short access);
void tstinit(void);
void tstbkg(void);

typedef void (*HDWIOFCNP)(int);
void hdwIoCycle(int cycle)
{
  /* read relevant hardware (here we just print something out) */
  printf("read hardware for cycle %d\n",cycle);
}
void onCycleTrigger1(int cycle,int cc,void *ref)
{
  printf("received cycle %d <%d>\n",cycle,cc);
```

# Release 4.2.0

- Odds and Ends
  - tine32.dll build without windows GUI references
    - i.e. no linkage to user32.dll, gdi32.dll, advapi32.dll
    - For pxi RT system with LabView
      - (Miha Vitorovic from cosylab)
  - More .NET customers !
    - REGAE experiment
      - 300kV diffractometer  (C# server)