Tip of the Month :

- How to use the Local and Central Archive/History Servers

TINE Archive Data
(a short review)

- **Central Archive Server**
  - Trends of 'registered' machine data stored *centrally*
  - Always on line (never removed or 'compressed')
  - 'Filtered' (remove as much of the 'haystack' from the 'needles' as possible)
  - Keep 'Points of Interest' to ensure the 'peaks' and 'valleys' appear in archive calls
- **Local History Subsystem**
  - Trends of 'registered' properties stored *locally* at the server
  - Short-term depth provides a 'ring-buffer' size
    - Data stored at archive polling interval
  - Long-term depth specifies storage range
    - Can also maintain a minimum disk space
    - 'Filtered' by tolerances only
  - Keep 'Points of Interest'
- **Event (*Post-mortem*) Archives**
  - Event triggers
    - Acquire and save according to trigger script.
    - Annotated (automatically + user comments)

Local History Data (notes)

- Configured via file
  - '*history.csv*' or '*fec.xml*'
- Configured via API
  - See: AppendHistoryInformation() (C-Lib)
    or: TEquipmentModule.addLocalHistoryRecord() (java)
- By default:
  - Use '*sequential*' history files
    - Appended
    - Fragmented (especially NTFS)
- Can specify 'standard' files
  - Use *Random-access* history files
    - Pre-allocated 'worst-case' files
    - Rotated (round-robin style)
  - Command line utility 'mkhstfiles' will create the 'standard' history file set.
  - **BIG** performance improvement in accessing local history data (especially on NTFS)

Accessing Archive Data

- Systematic details (*meta properties*)
  - <*property*>.HIST (or <*property*>.HST)
    - Returns:
      - Array of value-timestamp *doublets* (e.g. FLTINT, DBLDBL)
        - Normally: status != 0 does not get stored !
      - Array of INTFLTINT (doocs alias TDS) (i.e. timestamp-value-status)
        - If stored as e.g. FLTINT (value-status) then a status value can be supplied
      - Array of CF_HISTORY types !
        - Used systematically (*not for ordinary users*!)
        - Can carry any other data type !
      - Number of points in interval: *if requested output is a single number type* !

I bet you didn't know this !

Accessing Archive Data (via CF_HISTORY)

- C-Lib: GetArchiveDataAsAny()

```
int GetArchivedDataAsAny ( char *     devsrv,
                           time_t     start,
                           time_t     stop,
                           HstHdr *   dataHdr,
                           BYTE *     data,
                           int        dataFmt,
                           char *     dataTag,
                           int *      num
                         )
```

Retrieves archive data as requested in the call.

This call retrieves archive data from the archiver requested in the call. This call retrieves an archived data set according to the data format given.

Parameters:

devsrv	[in] must be the keyword-appended full device server name for which the archive data is desired.
start	[in] is the start time input (expressed as a UNIX timestamp) for which the archive data are desired.
stop	[in] is the end time input (expressed as a UNIX timestamp) for which the archive data are desired.
dataHdr	[out] is a pointer to an array to hold the history header information. This is an array of HstHdr objects containing a TINE timestamp (UTC double), a system data stamp (32-bit integer) and the user data stamp (32-bit integer) in one-to-one correspondence with the data array returned.
data	[out] is a pointer to an array of data objects to receive the archive data. This should an array of the desired data format (and large enough to hold the requested data).
dataFmt	[in] is the TINE data format code of the requested data. If this doesn't match the stored format, an attempt will be made to reformat the data. However this will not always be possible and could lead to an error.
dataTag	[in] is the TINE tagged structure tag to be used if the stored data is a TINE tagged structure. If the stored data is not a structure, this parameter is ignored.
num	[in/out] is a pointer to an integer giving (as input) the size of the data buffer which is to receive the archive data, and (as ouput) which contains the amount of archive data actually returned by the call.

Returns:

0 if successful, otherwise a TINE completion code which can be interpreted by a call to **GetLastLinkError( )**.

See also:

GetArchivedDataAsFloat( ), **GetArchivedData( )**, **GetArchivedDataAsText( )**

References **DTYPE::dArrayLength**, **DTYPE::data**, **DTYPE::dFormat**, **DTYPE::dTag**, **ExecLinkEx( )**, **DUNION::ulptr**, and **DUNION::vptr**.

Accessing Archive Data (via CF_HISTORY)

- Java: Thistory.getArchivedData()

```
double stpt = ((double) System.currentTimeMillis()) / 1000;
double srtt = stpt - 24 * 60 * 60;
HISTORY[] hstarr = new HISTORY[100];
for (int i=0; i<100; i++)
{
  hstarr[i] = new HISTORY(new TDataType(new FLTINT[1]));
}
int npts = THistory.getArchivedData("PETRA", "VAC.TSP","TestFloat","SOL21.8",
                                    0,srtt, stpt, hstarr,500);
TDataType rdt;
HISTORY h;
for (int i=0; i<npts; i++)
{
  h = hstarr[i];
  rdt = h.getDataObject();
  System.out.print(""+rdt.getDataTimeStamp() + " : "+ rdt.toString());
}
```

Accessing Archive Data

- Meta Property Input:
  - No input =>
    - *stop* = *now*
    - *start* determined by output data size
  - Up to 4 parameters:
    - *start* time (UTC) (default: given by output size)
    - *stop* time (UTC) (default: now)
    - *array index* (e.g. trace or spectrum array) (default: 0)
    - *sampling raster* (default: 0 => determined by output size and time range)

Accessing Archive Data

- '*normal viewing*': what do the archive viewers do?
  - THistory.getArchiveData() calls
    - Take a dimensioned array as output argument
      - determines requested output size (typically 2000)
    - Query 'number of points' from 2 sources (local and central archive)
      - Use source with fewest points > 500
    - Display and use 'optical zooming'
      - Any zoom reacquires data for new time range

Accessing Archive Data

- What if I want ALL data over a range?
  - Method #1:
    - send a sampling raster = 1
    - data buffer full => start again with timestamp of last buffer entry + 1
  - Method #2 :
    - first ask for number of points
    - then provide a buffer big enough and make a single call.

Accessing Archive Data

- *Snapshots* (details)
  - <*property*>.HIST@ (or <*property*>.HST@)
    - Returns the record (value or array set) at the *specified timestamp*.
      - i.e. nearest time equal to or more recent than requested time.
    - Returned *timestamp* is the *timestamp* of the data retrieved.

Accessing Archive Data

- How to display '*movies*'
  - Useful when the archive record is an *array* (either multi-channel or spectrum)
  - Method #1
    - get a trend over a time range (index = 0 or first device)
      - provides the *timestamps* of the stored data !
    - acquire and display snapshots at those *timestamps*
  - Method #2
    - Start at beginning and acquire first snapshot
    - Use data *timestamp + 1* to acquire next snapshot
    - Repeat until stop time is reached

Accessing Archive Data

- Related Meta Properties:
  - <*property*>.ARCH (or .ARC or .AR)
  - <*property*>.ARCH@
    - Redirects call to central archiver !
- Tips:
  - Try to avoid using the meta-properties yourself !
  - Use the utility routines
    - C-Lib: GetArchivedData() routines
    - Java: THistory.getArchivedData() methods