TINE Collaboration Meeting 2012

If your software project team can eat more than two pizzas, then it is too large!

Issues

- Release 5.0 Requirements
- Properties and Devices
- Security
- DOOCS compatibility
- Supported Platforms and APIs (including Web)
- Central Services
- *Standard* servers
- Video system
- Peripheral Applications (watchdogs, etc.)
- Diagnostics and logging
- CDI, TICOM, + other low-level interfaces
- Documentation and Forums
- Distribution / Repositories

Release 5.0 Requirements

- protocol headers
  - new protocol headers => new major release !
- string lengths
- contract coercion
- error codes
- supported data types
- performance criteria
- redirection / server groups
- hot-swapping / fail-over / redundancy
- configuration database
- ‘hooks’ to additional services

Protocol Headers

- all client requests begin with a packet header:

```
/**
 * \internal
 * \brief The top level header which defines the incoming data packet.
 *
 * \todo split totalSize away from the header.  This will involve a good look
 * at the routines in swaplib.c.  The advantage of doing this is primarily to
 * deal with a PktHdr with fields on a 4-byte boundary.  The first two
 * bytes of what has come in doesn't need to be a part of this header. BUT,
 * splitting this off from the header has low priority.
 *
 */
typedef struct /* struct is passed over the net */
{
  UINT16 totalSize; /**< total packet size in bytes (must match the bytes read) */
  char userName[USERNAME_SIZE]; /**< caller name */
  short tineProtocol; /**< tine protocol level (modern: 6 for contracts, 4 for globals) */
  UINT16 number; /**< tine version number (contracts) or number of incoming keywords (globals) */
} PktHdr;
#define PKTHDR_SIZE  22
#define SUB_PROT_LOCATION 18

#define LNKTBL_HASH_SIZE 211

#define REVISIONID(m,r) ((m)*256 + (r))

/**
 * \internal
 * \brief callback prototypes
 */
typedef void (*TCBFCNP)(int,int);
typedef void (*XTCBFCNP)(int,int,void *);
```

And it’s been like this since the Isolde days !

Protocol Headers: packet header

- 'tineProtocol' field tells the server which tine protocol the client wants to speak!
  - A legacy server will respond with 'illegal_protocol' if the requested level is too high (or too low).
    - e.g.#1: release 4 client contacts release 3 server
      - client receive 'illegal_protocol' and tries again with release 3 headers
      - everyone is happy ! ☺
    - e.g. #2: release 3 client contacts release 4 server
      - server is willing to work with release 3 headers
      - everyone is happy ! ☺
  - *Note: a release 4 server is NOT willing to use release 2 headers !*

Protocol Headers: packet header

- Points:
  - In order for backward/forward compatibility to work, the 'tineProtocol' field must always be in the same spot in the packet header !
  - So 'user name' is always *16 characters* or less !
    - Is this an issue ? (see 'String lengths' later).
    - (*actually not really true: just need byte 18 to point to the protocol level*)
  - It's probably time to split off the 'totalSize' field from the header definition (or not?).
    - totalSize is an unsigned short => maximum 65535 bytes
    - tine does its own packet reassembly => *not a problem*!
  - This also happens in Java, but the incoming/outgoing byte streams are mapped into fields in a class
    - *Java does not have structures*
    - *Java does not have unsigned integers*

- Java:

```
public TReqHdr(byte[] data,int off, int len)
{ // prepare incoming ...
  int bytesread = 0;
  try
  {
    ByteArrayInputStream dinBuffer = new ByteArrayInputStream(data,off,len);
    DataInputStream ds = new DataInputStream(dinBuffer);
    // overall message length:
    totalSizeInBytes = Swap.Short(ds.readShort());
    bytesread = 2;
    Arrays.fill(bstr, (byte)0);
    while (bytesread < hdrSizeInBytes)
    {
      ds.read(bstr,0,16);
      username = (new String(bstr)).trim();
      tineProtocol = Swap.Short(ds.readShort());
      revisionId = Swap.Short(ds.readShort());
      bytesread += 20;
    }
    ds.close();
    dinBuffer.close();
  }
  catch (IOException e)
  {
    e.printStackTrace();
    MsgLog.log("TReqHdr",e.toString(),TErrorList.code_failure,e,0);
```

signed short

Protocol Headers: packet header

Maximum Transport Unit

- MTUs
  - Response (server -> client)
    - *settable*: 512 -> 64 K
    - *default* : 1472
  - Request (client -> server)
    - settable only in C and only at compile time
    - default: 1472
    - some builds (NIOS) had 1000 bytes
    - note: client requests *rarely* send large data sets to server !
    - => *also allow this to be settable* !
      - (and fix the java code)

- Subscription Header from client
  - can be *packed* (*multiple requests in a packet*)
  - any packet reassembly (due to long input data) **must** recognize the contract !

```
/**
 * \internal
 * \brief Contract header struct
 *
 * struct is passed over the net : tineProtocol 6 (in)
 */
typedef struct subscription
{
  UINT16     msgsize;         /**< total message size in bytes */
  UINT16     extsize;         /**< extended string space */
  UINT16     mtu;             /**< maximum transport unit for incoming data */
  UINT16     numblks;         /**< total number of message blocks */
  UINT16     blknum;          /**< block number of this message */
  UINT16     id;              /**< block id */
  short      mode;            /**< requested transfer mode (e.g. CM_TIMER, CM_DATACHANGE) */
  short      blkid;           /**< (incremented) block id */
  UINT32     pollingInterval;/**< requested polling interval in msec */
  UINT32     starttime;       /**< UTC starttime of the contract (sent from client) */
  CONTRACT contract;          /**< contract request */
} SubInfoPkt;

#define SUBINFO_SIZE (8*2+2*4+CONTRACT_SIZE)
#define SUB_MODE_LOCATION (PKTHDR_SIZE + 8)
```

212 bytes

Protocol Headers: subscription

Protocol Headers: CONTRACT

- The contract structure:
  - together with input data uniquely specifies the *call* !
  - EqmDeviceName ends with '&' => use extended string space.

```
/**
 * \internal
 * \brief Contract input header for protocol level 6
 *
 * structure is passed over the net
 *
 */
typedef struct
{
  char   EqmProperty[PROPERTY_NAME_SIZE]; /**< requested property */
  char   EqmDeviceName[DEVICE_NAME_SIZE]; /**< requested device */
  char   EqmName[EQM_NAME_SIZE];          /**< requested local eqm name */
  UINT32 EqmSizeIn;                       /**< input data size */
  UINT32 EqmSizeOut;                      /**< output data size */
  BYTE   hEqmName;                        /**< handle to eqm name (unused) */
  BYTE   EqmAccess;                       /**< requested access (CA_READ etc) */
  BYTE   EqmFormatIn;                     /**< input data format */
  BYTE   EqmFormatOut;                    /**< output data format */
  char   strTagIn[TAG_NAME_SIZE];         /**< input data tag */
  char   strTagOut[TAG_NAME_SIZE];        /**< output data tag */
} CONTRACT;
#define CONTRACT_SIZE (PROPERTY_NAME_SIZE+DEVICE_NAME_SIZE+EQM_NAME_SIZE+4*2+4*1+2*TAG_NAME_SIZE)
```

188 bytes

Protocol Headers: the Request

- A request from a client contains:
  - PktHdr (the incoming packet header)
    - gives total number of bytes in the request
  - N x the following:
    - where N = number of packed requests
    - SubInfoPkt (subscription request)
    - *extended string space* if any (long device names)
    - *input data*
  - keep looping until the total number of bytes have been read.
    - most of the time : N = 1

- producer header (what the client sees)

```
/**
 * \internal
 * \brief producer header struct (reply to client)
 *
 * struct is passed over the net : tineProtocol 6 (out)
 *
 */
typedef struct
{
  UINT16 msgsize;         /**< message size in bytes */
  UINT16 subId;           /**< supplied by the consumer upon registration (con tbl id) */
  UINT16 CompletionCode;  /**< return completion code from the eqm */
  UINT16 numblks;         /**< total number of message blocks */
  UINT16 blknum;          /**< block number of this message */
  UINT16 mtu;             /**< maximum transport unit */
  UINT16 EqmFormat;       /**< format of returned data */
  UINT16 counter;         /**< subscription counter (how much is left) */
  UINT16 tineProtocol;    /**< tine protocol level */
  UINT16 xferReason;      /**< transfer reason flag (one of the CX_ codes above) */
  UINT32 ClnStarttime;    /**< supplied by the consumer upon registration (@ byte 20)*/
  UINT16 stssize;         /**< size of error/status string space */
  UINT16 stscode;         /**< access status code */
  UINT32 timestamp;       /**< data timestamp */
  UINT32 timestampUSec;   /**< usec fraction of data timestamp */
  UINT32 userstamp;       /**< application setable data stamp */
  UINT32 sysstamp;        /**< systematic data stamp */
} PrdrHdr;

#define PHDR_SIZE (12*2+5*4)
#define PMTU(m,np,tp)  (DGMTU(m,np,tp) - ((tp) == 5 ? PHDR5_SIZE : PHDR_SIZE) - sizeof(UINT16))
#define PHDR_STTM_OFFSET 20
#define PHDR_PROT_OFFSET 16
#define PHDR_STSSIZ_OFFSET 24
```

44 bytes

Protocol Headers: the Reply

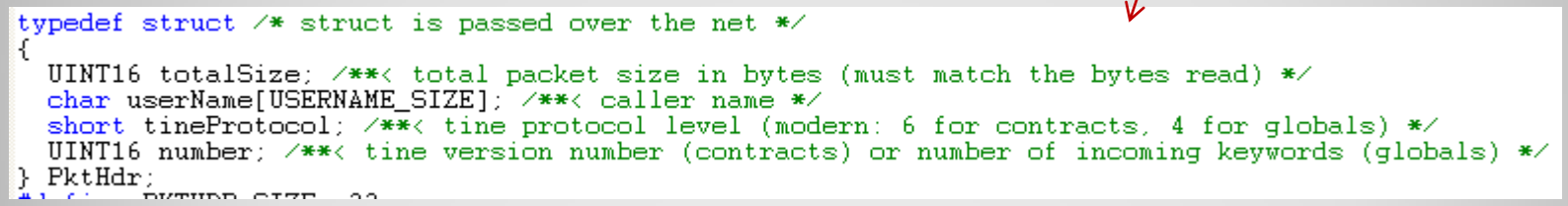
Protocol Headers: the Reply

- A reply from a server contains:
  - totalSizeInBytes as int16
    - (n.b. for CF_STREAM: as int32)
  - N x the following:
    - PrdrHdr (returned producer header)
    - *associated data* if any
    - *status string* if any (up to 192 bytes)
  - Keep looping until totalSizeInBytes has been handled.
  - Note: packet reassembly vs. packed responses
    - large data set: N = 1 and many packets
    - but: *several contracts returning '1 float' can be packed!*

Protocol Headers: the Reply

- Some examples …
  - *a write command* :
    - no returned data !
    - success => just the PrdrHdr
    - failure => PrdrHdr + status string
  - *a read request* (success) :
    - PrdrHdr
    - the data
  - *a read request* (failure) :
    - PrdrHdr
    - status string
  - *a read request* (status != 0 + CE_SENDDATA) :
    - PrdrHdr
    - the data
    - status string

- Additional information in request (PktHdr)?
  - caller pid ? why ?
  - others IDs ?
    - client application '*type*' ?
      - script, MatLab, middle layer, GUI, etc.
      - How to determine this ?
  - *endianness* flag ?
    - or stick with little endian ?
  - character encoding flag ?
    - or use UTF8, stick with ascii ?
  - anything else ?
  - reserved space ?



```
typedef struct /* struct is passed over the net */
{
  UINT16 totalSize; /**< total packet size in bytes (must match the bytes read) */
  char userName[USERNAME_SIZE]; /**< caller name */
  short tineProtocol; /**< tine protocol level (modern: 6 for contracts, 4 for globals) */
  UINT16 number; /**< tine version number (contracts) or number of incoming keywords (globals) */
} PktHdr;
```

Protocol Headers: release 5.0

- Additional Information in Subscription ?
  - use contract tag/id for reassembly packets ?
    - instead of repeating the CONTRACT
    - saves repeating 188 bytes of the **mtu** size
    - *n.b.* only need reassembly when *sending* a large data input set.
    - not worth the bother ?
  - anything else ?

Protocol Headers: release 5.0

- Additional Information in CONTRACT ?
  - supply input/output data sizeInBytes ?
    - data size + format do not always *uniquely* determine size in bytes for some data types !
      - CF_STRING, CF_AIMAGE + other adjustable length data types (and structures that contain them)!
      - the data 'tag' is currently (mis-)used for this info.
  - *Maybe*: sizeInBytes and sizeInElements ?
  - extended string space for property names ?

Protocol Headers: release 5.0

- Additional information in reply header ?
  - as in CONTRACT: need number of elements !
  - anything else ?

```
typedef struct
{
  UINT16 msgsize;          /**< message size in bytes */
  UINT16 subId;            /**< supplied by the consumer upon registration (con tbl id) */
  UINT16 CompletionCode;   /**< return completion code from the eqm */
  UINT16 numblks;          /**< total number of message blocks */
  UINT16 blknum;           /**< block number of this message */
  UINT16 mtu;              /**< maximum transport unit */
  UINT16 EqmFormat;        /**< format of returned data */
  UINT16 counter;          /**< subscription counter (how much is left) */
  UINT16 tineProtocol;     /**< tine protocol level */
  UINT16 xferReason;       /**< transfer reason flag (one of the CX_ codes above) */
  UINT32 ClnStarttime;     /**< supplied by the consumer upon registration (@ byte 20)*/
  UINT16 stssize;          /**< size of error/status string space */
  UINT16 stscode;          /**< access status code */
  UINT32 timestamp;        /**< data timestamp */
  UINT32 timestampUSec;    /**< usec fraction of data timestamp */
  UINT32 userstamp;        /**< application setable data stamp */
  UINT32 sysstamp;         /**< systematic data stamp */
} PrdrHdr;
```

Protocol Headers: release 5.0

Release 5.0: string lengths

- Relevant string lengths in these headers:
  - DEVICE_NAME_SIZE: 64 bytes
    - the registered device name length!
      - queries usually ask for a list of NAME64 items
    - BUT: can use extended space (up to 1024 bytes)
      - e.g. requesting a 'list' of names as in "cdiDev1,cdiDev2,cdiDev3,…" or "motor1,motor3,motor5,…"
      - Using 'DeviceName' as a free parameter to supply e.g. a file path
  - PROPERTY_NAME_SIZE: 64 bytes
    - the registered property name length!
    - no extended string space option
  - EQM_NAME_SIZE: 8 bytes
    - the 'local' equipment module name
    - historically only 6 characters (NODAL!) have even been used.
  - TAG_NAME_SIZE: 16 bytes
    - For tagged structures, bitfields, + some 'other' cases
  - USERNAME_SIZE: 16 bytes
    - parallels the FECNAME_SIZE
    - has always been enough, BUT:
      - Windows users name can be 20 characters

Are these OK ?

- FEC address structure:
  - a FEC manages 1 or more EQM
  - an EQM is the internal representation of a device server

```
/* these macro re-definitions of the IPX address fields will make sections of the code easier to read */
#define MMFhwnd IPXImmediateAddress
#define PIPEsck IPXNetwork
#define PIPEerr IPXNode[0]
/**
 * \internal
 * \brief structure which contains the FEC address information
 *
 * The TINE library uses this structure for internal caches.
 * Sent to and received from the ENS
 *
 * \note struct is passed over the net (64 bytes)
 *
 */
typedef struct
{
  char fecName[FEC_NAME_SIZE]; /**< FEC name */
  BYTE IPXNetwork[4];          /**< IPX addr info (or internal) */
  BYTE IPXNode[6];             /**< IPX addr info (or internal)  */
  BYTE IPXImmediateAddress[6]; /**< IPX addr info (or internal)  */
  char IP[ADDR_SIZE];          /**< IP address as string */
  char IPh_addr[4];            /**< IP haddr (byte representation) */
  SINT32 portOffset;           /**< port offset (applied to all listening ports) */
  SINT32 inetProtocol;         /**< inet protocol (UDP, TCP, IPX, MMF, PIPE, etc.) */
  SINT32 tineProtocol;         /**< tine protocol level (highest level accepted) */
} FecDataStruct;
#define FECDATASTRUCT_SIZE (FEC_NAME_SIZE+16+16+4+3*4)
```

Release 5.0: addresses

- EQM address structure
  - principal addressable information:

```
/**
 * \internal
 * \brief structure which is used to map a EqmContext/ExportName to a FecName/EqmName.
 *
 * The client uses EqmContext and Export name to address the FEC.
 * TINE internally uses the FecName and the Equipment name.
 *
 * The TINE library uses this structure for internal caches.
 * Sent to and received from the ENS
 *
 * \note struct is passed over the net (104 bytes)
 *
 */
typedef struct
{
  char  FecName[FEC_NAME_SIZE];          /**< FEC Name */
  char  SubSystem[SUBSYSTEM_NAME_SIZE]; /**< sub-system name */
  char  ExportName[EXPORT_NAME_SIZE];    /**< Export Name */
  char  EqmName[EQM_NAME_SIZE];          /**< Equipment name */
  char  EqmContext[CONTEXT_NAME_SIZE];   /**< EqmContext */
} ExpDataStruct;
```

  - additional FEC information:
    (not important for client-server communication)

```
typedef struct /* new */
{
  char os[FEC_OS_SIZE];
  char desc[FEC_DESC_SIZE];
  char loc[FEC_LOCATION_SIZE];
  char ver[FEC_VERSION_SIZE];
  char hdw[FEC_HDW_SIZE];
  char resp[FEC_RESP_SIZE];
} FecInfoStruct;
```

Release 5.0: addresses

Release 5.0: string lengths

- Relevant string lengths in these structures
  - FEC_NAME_SIZE: 16 bytes
    - parallels USERNAME_SIZE
  - ADDR_SIZE: 16 bytes
    - semi-redundant if IP addr as byte representation also present
  - SUBSYSTEM_NAME_SIZE: 16 bytes
  - CONTEXT_NAME_SIZE: 32 bytes
    - n.b. does NOT appear in protocol headers!
  - EXPORT_NAME_SIZE: 32 bytes
    - n.b. does NOT appear in protocol headers!

Release 5.0: IPv6 support

- **IPv4**:
  - 4 bytes
  - string representation: e.g. “131.169.151.47” (16 chars)
- **IPv6**:
  - 16 bytes
  - string representation: e.g. “2001:0db8:85a3:0000:0000:8a2e:0370:7334” (40 chars)
  - 64-bit network prefix + 64-bit interface id
- **IPv4** to **IPv6** mapping
  - hybrid dual-stack (hybrid sockets)
  - 80 ‘0’ bits + 16 ‘1’ bits + remaining 32 IPv4 bits
  - “0000:0000:0000:0000:0000:ffff:83a9:972f” OR
  - “::ffff:83a9:972f” OR
  - “::ffff:131.169.151.47”
- Tunneling
  - IPv4 only hosts communicating with IPv6 only hosts

Release 5.0: IPv6 support

- Proposed FEC address structure:
  - no real IPX support
  - inetProtocol:
    - UDP, TCP, …
    - UCPv6, TCPv6, …
  - 96 bytes (was 64 bytes)

```
/* definitions for future release: */
typedef struct
{
  BYTE IpxNet[4];
  BYTE IpxNode[6];
  BYTE IpxImmAdr[6];
} IpxAdr;
typedef union
{ /* 16-bytes */
  BYTE ipv6[16];
  BYTE ipv4[4];
  IpxAdr ipx;
} FecNetAdr;
typedef union
{ /* force 8-byte size */
  HWND wndMmf;
  SOCKET sckPipe;
  BYTE reserved[8];
} FecLclAdr;
/**
 * \internal
 * \brief structure which contains the FEC address information
 *
 * The TINE library uses this structure for internal caches.
 *
 * \note struct is passed over the net (96 bytes)
 *
 */
typedef struct
{
  char fecName[FEC_NAME_SIZE]; /**< FEC name */
  FecNetAdr netAdr;            /**< network addr info */
  FecLclAdr lclAdr;            /**< local addr info (pipe, mmf) */
  char strAdr[40];             /**< addr info as string */
  SINT32 portOffset;           /**< port offset */
  SINT32 inetProtocol;         /**< inet protocol type */
  SINT32 tineProtocol;         /**< tine protocol level */
  SINT32 reserved;             /**< reserved integer */
} FecAddrStruct;
#define FECADDRSTRUCT_SIZE (FEC_NAME_SIZE+16+8+40+4*4) /* 96 bytes */
```

Release 5.0: IPv6 support

- Considerations:
  - how much *hybrid* support ?
    - a server listens on **IPv6** *OR* **IPv4** sockets but not both, etc.
    - use ‘hybrid sockets’ ?
  - this will also affect the *ipnets* access lists
  - how best to be **IPv6** ready ?

Release 5.0: character sets

- Currently:
  - C-Lib and Java Lib transfer ascii 1-byte chars over the net.
  - Java Lib API uses unicode char strings.
  - C-Lib API uses ascii char strings.
    - BUT VB 6, .NET, MatLab, etc. wrappers use unicode.
  - most applications are unaware that they are NOT using unicode unless they try to pass e.g. Japanese characters.
  - switch to UTF-8 ?

- Currently all strings are transferred as ascii characters (1 char = 1 byte)
- UTF8:
  - represent all characters in unicode
  - variable width encoding
  - backward compatible with ascii
  - no endianness or byte-order mark problems

Bits	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+007F	0xxxxxxx					
11	U+07FF	110xxxxx	10xxxxxx				
16	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
21	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

acsii 0 – 127: covers English

Release 5.0: character sets

Release 5.0: UTF-8 support

- encode/decode all string data as UTF-8 ?
  - also: exposed *names* such as Device Name, Property Name, Context, etc.
  - file i/o: convert to locale settings?
  - transferred data
- existing APIs remain unchanged.

- Servers can steer *inelegant* client requests in the right direction.
  - BUT the 'steering' information needs to be registered !
  - SetMinimumAllowedPollingInterval()
  - RegisterPropertyInformation(,,, access, array_type, ,,redirection)
    - array_type = CA_CHANNEL
      - enforce MCA acquisition
    - access = CA_NETWORK
      - enforce multicast access
      - also blocks synchronous calls
    - access = CA_STATIC
      - block monitors
    - access = CA_NOSYNC
      - block synchronous polling
      - automatic start of client side listener ?
    - redirection != NULL
      - requests to this property go to another server
    - data type = registered structure
      - access of a structure field will return entire structure
  - RegisterMultiChannelGroupDevice()
    - alternative to CA_CHANNEL for strict OO devices
  - RedirectDeviceName(,, redirection)
    - redirection != NULL
      - requests to this device go to another server

Release 5.0: contract coercion

- Relevant '*hand-shaking*' status codes:
  - *caller should never see these* !
  - FEC/Server steering:
    - invalid_protocol < establish communication protocol
    - invalid_interval < establish polling interval
  - property steering:
    - get_subscription_id < listen for multicasts
    - property_is_mca < provide index to MCA
    - reset_mca_property < when MCA elements change
    - information_static < stops polling of static data
    - server_redirection < redirect request
    - async_access_required < block synchronous acquisition
    - mcast_access_required < require multicast access
    - has_structure_tag < field has underlying structure
  - device steering:
    - data_not_local < wildcard device is not local
    - server_redirection < redirect request

Release 5.0: contract coercion

- What are we trying to *avoid*?
  - sending same data set to a long list of clients !
    - CA_NETWORK
  - inefficient/counter productive polling intervals
    - property is being scheduled at a high rate
    - set minimum polling interval
  - single element acquisition of a known multi-channel array
    - CA_CHANNEL
  - single field acquisition of a known structure
  - polling/monitoring static information
    - CA_STATIC
    - e.g. the units are “Amperes” and they aren’t going to change!
  - synchronous polling of something that should be monitored
    - CA_NOSYNC
  - anything else ?

Release 5.0: contract coercion

Release 5.0: error codes

- error codes/status codes < 512 are deemed '*systematic*'
  - not all 'errors' or 'exceptions' !
  - some are *ancient* (date back to the Isolde days)
    - need the 'tdc_' prefix ?
  - some are for *handshaking*
    - *n.b.* error codes from a dispatch routine run thru a validator !
  - some are informational
    - CE_SENDDATA
      - e.g. has_query_function
  - some indicate '*link*' or network errors
  - some indicate '*call*' errors
  - do we need more structure in this?
  - prune unnecessary codes ?
  - any obvious missing codes ?

Release 5.0: data types

- Supported data types
  - All the '*primitives*'
    - CF_BYTE 1 byte
      - byte (some 32-bit C, VB6, java, .NET); also Int8 (.NET)
      - Unsigned byte (.NET); also UInt8 (.NET)
    - CF_CHAR (CF_TEXT) 1 byte
      - char (C without #define UNICODE, .NET with ansi encoding)
      - byte (java, VB6)
    - CF_INT16 (CF_SHORT) 2 bytes
      - int (VB6, 16-bit C)
      - short (not MatLab); also Int16 (.NET)
      - unsigned short (not java, VB6, MatLab); also UInt16 (.NET)
    - CF_INT32 (CF_LONG) 4 bytes
      - long (VB6, non 64-bit C, MatLab)
      - int (32-bit, 64-bit C, java, Labview); also Int32 (.NET)
      - unsigned int, long (32-bit, 64-bit C, Labview); also UInt32 (.NET)
    - CF_INT64 (CF_DLONG) 8 bytes
      - long long (UNIX 32-bit C)
      - _int64 (WINDOWS 32-bit C)
      - long (64-bit C, java, .NET); also Int64 (.NET)
      - unsigned long (64-bit C, .NET); also UInt64 (.NET)
    - CF_FLOAT 4 bytes
      - float (C, java, .NET)
      - single (VB6, LabView)
    - CF_DOUBLE 8 bytes
      - double (everybody!)

Release 5.0: data types

- Primitives
  - do we need explicit 'unsigned' definitions?
    - it's '*only*' a matter of interpretation at the end points, BUT you have to know a priori how to interpret!
  - or: just '*do a java*' and claim everything is signed
    - and leave the developer to his tricks…
  - *note:* 'STRING' is NOT a primitive !

Release 5.0: data types

- String types
  - CF_CHAR (CF_TEXT)
    - (an array of) 1-byte characters
    - i.e. a string
  - CF_NAME8, CF_NAME16, CF_NAME32, …CF_NAME64
    - (an array of) fixed-length (i.e. fixed capacity) strings
    - very good for querying lists
    - very efficient to traverse
  - CF_STRING
    - (an array of) mutable strings
    - *in C this corresponds to an array of pointers !*
  - CF_KEYVALUE
    - (from the doocs world)
    - parallels CF_STRING
    - (an array of) mutable strings of the form “key: value”
  - CF_XML
    - (from the doocs world)
    - parallels CF_TEXT

Release 5.0: data types

- Compound Data Types
  - doublets
    - e.g. CF_LTINT, CF_DBLDBL, CF_INTINT, CF_NAME32I, etc.
  - triplets
    - e.g. CF_FLTFLTINT, CF_NAME64DBLDBL, etc.
  - quads
    - e.g. CF_ADDRESS, CF_FILTER, etc.
  - special
    - e.g. CF_SPECTRUM, CF_IMAGE, etc.
    - Header + designated length of some other type (each element in an array of these has the same length)
  - adjustable length
    - e.g. CF_ASPECTRUM, CF_AIMAGE, etc.
    - Header + adjustable length of some other type (each element in an array of these can have a different length)
  - systematic
    - e.g. CF_HISTORY
  - (almost) complete overlap with DOOCS data types
  - *remove deprecated types !*
    - e.g. CF_DBLINT only ever made sense on MSDOS

Release 5.0: data types

- tagged structures
  - can contain any other data type
    - except CF_HISTORY
    - can also contain 'adjustable' types
  - can be nested
  - best practice: use primitives and don't nest too deeply
    - .NET:
      - has structures
      - if structure is 'blitable' (all primitives) then the block of memory is easily accessible and handled more efficiently.
    - MatLab:
      - Essentially composed of (arrays of) char, long, and double
  - (first order) fields can be read (but not written) independently
    - BUT entire structure is always delivered.
    - do we need to read nested fields independently ?
  - 'tag' and 'field' names are limited to 16 characters
    - is this a problem ?
    - note: accessing a field => request <property>.<field>
    - restrict <property> length to 64 – 16 characters if data type = CF_STRUCT ?

Release 5.0: data types

- Bitfields
  - Applies only to ‘integer’ types
    - CF_BITFIELD8, CF_BITFIELD16, CF_BITFIELD32, …
  - can be used to enumerate bits
  - can also give names to ‘fields’ of bits
  - can read any field independently
  - WRITE commands pass the data sent to the dispatch handler as is.
  - How to WRITE bits (bit fields) independently?
    - Somehow pass the ‘field’ or field mask to the dispatch?
  - Note: the .BIT.x *meta*-properties have some overlap here.

- Tweaking Performance
  - Quality of Service
    - UDP, TCP, STREAM, PIPEs and MMFs
  - threads
    - priorities
  - default settings
    - LAZY vs. EAGER scheduling
    - flow control parameters for UDP
    - thread priorities and synchronization
    - deadbands, timeouts
    - lingering canceled contracts
    - default table lengths
      - client, contract, connection tables …
      - resources_exhausted ?
      - use ArrayLists in java after all ?
  - bottlenecks ?
    - eqm dispatch is synchronized
      - but can run on separate thread if needed (other calls would get 'operation_busy' rather than a 'link_timeout').
      - can also return 'not_ready'
      - Can optionally synchronize with the background dispatch.
  - Other issues?

Release 5.0: performance issues

Release 5.0: redirection/groups

- Redirection:
  - from any /context/server/device[property] to any other /context/server/device[property].
    - requires 'status string' to be up to 192 bytes.
    - no '*long*' device name allowed here!
- Group Equipment Name Server (GENS)
  - redirects the device entries in its database to the appropriate target server.
  - needs device 'metric' if the device order is important
  - Can apply device name pre- and post-fixes to avoid device name collisions if necessary.
- Archive System
  - central archive redirects back to device server for 'local' history information
  - device server redirects to central archive of '.ARCH' meta-properties
- redirection issues ?

Release 5.0: hot swapping

- hot swapping devices
  - use
    - RegisterDeviceName(), AssignNumDevices(), SetSizeDeviceCapacity()
    - should call ResetMultiChannelProperty() if hot swapping a device within an MCA
  - any issues ?
- adding/editing settings *on-the-fly*
  - local histories
  - alarm watches
  - units, max/min settings
    - additions/edits are currently '*volatile*'.
    - save the changes ?
      - there's not always a config file!

Release 5.0: failover

- Software failover of device server
  - *2 servers with identical functionality*
    - e.g. /PETRA/Idc.OR08 and /PETRA/Idc.OR19
    - one is declared master
      - Registers itself a 2nd time with a 'common' name
      - e.g. as /PETRA/Idc
    - one is declared slave
      - monitors the master
  - failure of master: triggers the slave to register as master
  - return of real master: should resume its role!
  - running client will experience a down time on the order of minutes.
  - '*best source*' scheme (vs. load balancing scheme).
  - Should this be configurable?
    - e.g. best source or load balancing

Release 5.0: configuration

- Configuration options
  - *Clients need to resolve addresses* !
    - without **ENS** (Equipment Name Server)
      - run in stand-alone mode (TINE_STANDALONE=TRUE)
        - use the 'dynamic' cache
        - explicit instruction NOT to use the ENS
      - use a 'local' database repository
        - TINE_HOME points to it
    - with **ENS**(es)
      - need to 'find' the **ENS**(es)
        - TINE_HOME -> cshosts.csv (contains known address)
        - TINE_ENS specifies it
        - DNS + 'tineens'
        - multicast: 'ENS where are you?'
        - assign via API
    - any other ideas ?

Release 5.0: configuration

- Configuration options
  - servers can register all info via API
    - and some do !
      - all doocs servers
      - epics2tine
      - tango2tine
      - etc.
  - else
    - FEC_HOME points to FEC database repository
    - can use xml as database (fec.xml) or .csv files.
    - can also use API registration as well.
  - but
    - save-and-restore only uses .csv files
    - and uses the FEC_HOME repository
  - any issues ?

- csv File configuration
  - **FEC_HOME** points to repository
  - fecid.csv found there : FEC name and port
    - issues with fecid.csv ?
  - each registered equipment module (EQM) has a subdirectory given by its process-'local' EQM name (6 characters)
    - e.g. ./BLMEQM
    - exports.csv : export (server) name, property info
    - devices.csv : device names + info
    - users.csv : allowed users (and groups)
    - history.csv : local history info
    - alarms.csv, almwatch.csv : alarm system info
    - ipnets.csv : allowed net addresses
    - + <*property*>-users.csv, <*device*>-users.csv, etc.
    - + save-and-restore files
  - backward compatibility:
    - also see if these files exist directly under FEC_HOME area !

Release 5.0: configuration

- exports.csv
  - spreadsheet like
    - focus on: exported property information
    - => no hierarchy !
    - some column-repeated information:
      - *context*, *export name*, *number devices*, etc.
    - max, min, units now parsed from description
      - Provide extra optional columns ?
        - MAX, MIN, UNITS, XMAX, XMIN, XUNITS ?
- fec.xml
  - is hierarchical
  - same tags as .csv columns
  - all info in 1 large file
    - a bit more cumbersome to auto-update
    - no possibility of e.g. inserting a history.csv file 'under' a server.

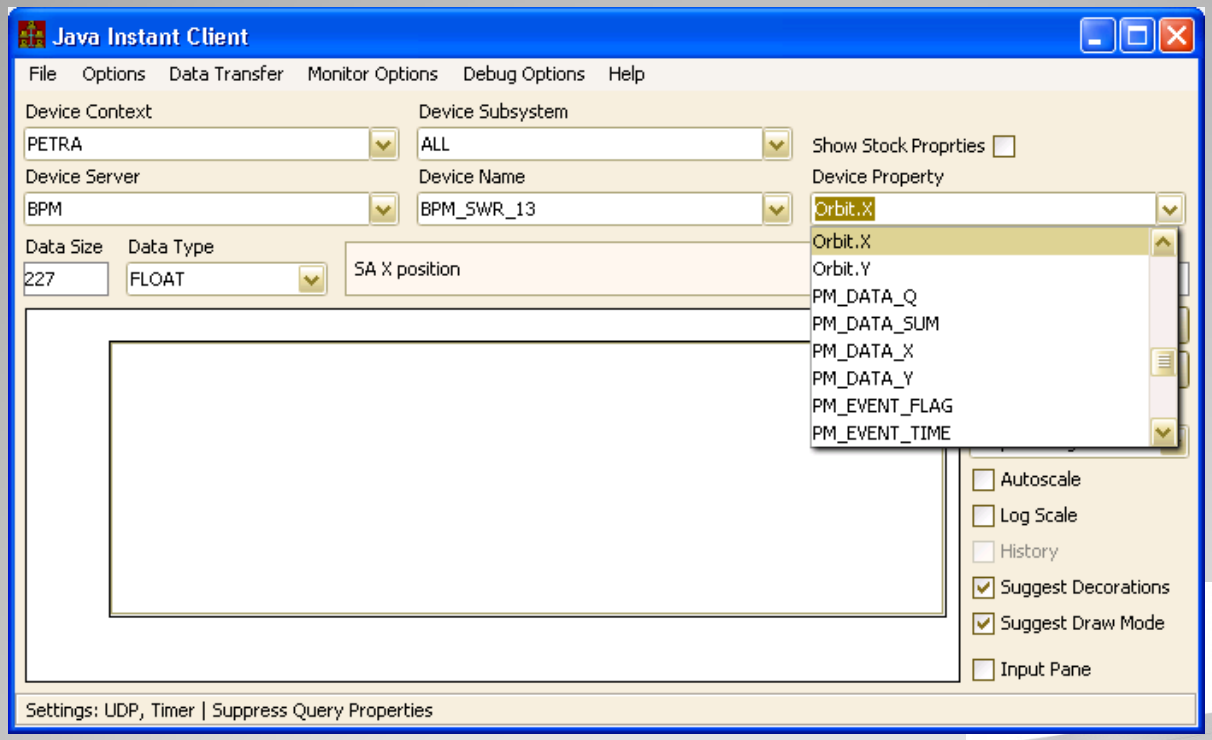
Release 5.0: configuration

Release 5.0: configuration

- dynamic cache location ?
  - use an environment variable ?
  - current defaults:
    - Win32: %SystemDrive%:\tine\cache
    - unix: /var/tmp/tine/cache
      - but this is cleared on reboot of host
      - check for /var/tine/cache (with o:rw)?

Release 5.0: hooks

- hooks to additional resources ?
  - currently: hook for external fd (sockets) sets
  - Others ?
    - e.g. a hook for a real-time delay : rtdelay() ?



Properties and Devices

Properties and Devices

- Properties
  - are *methods* !
  - provide the essential point of contact to the equipment module dispatch handler
    - *no property -> no dispatch* !
  - can have *access control lists*
  - have *meta information*
    - canonical data size and type
    - units
    - max and min settings
    - etc.

- Properties

```
/**
 * \internal
 * \brief Structure used to hold exported property information.
 */
typedef struct ExportPropertyList
{
  char prpName[PROPERTY_NAME_SIZE];/**< property name */
  char prpAlias[PROPERTY_NAME_SIZE];/**< property alias name */
  SINT32 prpId;              /**< property id */
  UINT32 prpSize;            /**< maximum supported array size */
  UINT32 prpSizeIn;          /**< maximum supported input array size */
  short prpFormat;           /**< format output data */
  short prpFormatIn;         /**< format input data */
  short prpAccessMode;       /**< bit 0: read access, bit 1: write access */
  short prpArrayType;        /**< property array type if array */
  short numRows;             /**< num rows if array */
  short rowSize;             /**< row size if array */
  NAME16 prpFormatTag;       /**< output data structure tag if non zero-length */
  NAME16 prpFormatTagIn;     /**< input data structure tag if non zero-length */
  PrpRedirBlk *prd;          /**< re-director if not NULL */
  NAME64 *devNames;          /**< device list if not NULL */
  int numDevices;            /**< number of devices in device list */
  AclInfoType aclLst;        /**< property specific acls */
  int hasExclusiveRead;      /**< flag == PRP_XREAD_NONE, _LOCK, _ALL */
  int runInSeparateThread;   /**< flag == TRUE -> eqm call in separate thread */
  char prpDescription[PROPERTY_DESC_SIZE];/**< short description of property */
  char *prpUrl;              /**< url to additional information (if not NULL) */
  PrpEgu egu;                /**< EGU information for display */
  PrpEgu xegu;               /**< x-axis EGU information for spectrum display */
  BYTE *prpBuf;              /**< deeply bound data buffer if not NULL */
  BYTE *srBuf;               /**< save/restore data buffer if not NULL */
  UINT32 srBufSiz;           /**< save/restore data buffer size */
  PRPSIG sigfcn;             /**< property signal function if registered */
  int sigmask;               /**< assigned signal mask to use in a signal function */
  void *sigref;              /**< assigned reference to use in a signal function */
  time_t mcaValidFloor;      /**< minimum client startime where mca indexing is valid*/
  struct ExportPropertyList *next;
} ExportPropertyListStruct;
```

```
typedef struct PrpEguStruct
{
  char units[16];
  float min;
  float max;
  BYTE graph;
  BYTE reserved[3];
} PrpEgu;
```

Properties and Devices

Properties and Devices

- Questions:
  - Do we want to distinguish between
    max and min 'display' settings
    and
    max and min 'set point' settings ?
  - Should there be an (optional) '*automatic*' out_of_range check if attempt to WRITE a value past the set points ?
  - Any other missing meta-attributes ?

Properties and Devices

- *meta* properties
  - property name + up to 4 char meta extension
    - e.g.
      - LossRates.HIST (history of property "LossRates")
      - Charge.EGU (engineering units of "Charge")
      - Orbit.X.NAM (associated channel names for "Orbit.X")
      - Trace.XMIN (x-axis min value for "Trace")
      - etc.
  - gated meta properties
    - e.g.
      - Status.BIT.3 (bit 3 of 'integer' property "Status")
      - Register.MASK.0x7 (value of "Register" masked by 0x07)
      - Status.Gate.0xae (binary output of "Status" gated against 0xae)
    - Coming soon:
      - Pressure.DMASK.3 (**MCA** returns those devices whose device mask is '3' – doocs SYS_MASK)
      - Pressure.DMASK.3.NAM (**MCA** device names whose device mask is 3)
      - Possible enumerations for the '3' ?
        - e.g. Pressure.DMASK.turbo

Properties and Devices

- Multi-Channel Array (**MCA**) properties
  - Required behavior
    - must supply an array of equal length and corresponding to either
      - 1) the registered device list
      - 2) an assigned device list
        - see AssignDeviceListToProperty()
      - 3) another registered property of the same name but with the meta-extension ".NAM".
    - *note:* using either 2) or 3) above automatically flags the server as having 'property query precedence' (i.e. a '*property server*' instead of a '*device server*').
    - must accept the contract's 'DeviceName' as the starting point in the **MCA** and return the number of elements requested.
      - often 1 element OR all elements starting at the beginning.
      - the dispatch can *wrap* past the end or *truncate* the call as desired.
  - can also make use of RegisterMultiChannelGroupDevice() if there is a hard device query precedence!

Properties and Devices

- Devices
  - may or may not refer to hardware devices
  - can have:
    - property lists
      - which of the registered properties are supported by this device ?
      - flags the server as having device query precedence.
    - access control lists
    - description
    - location
    - Alarm lists
    - mask (*doocs SYS_MASK*)
    - Z (longitudinal) position

Properties and Devices

- Any open issues?
  - wildcard support ?
    - both DeviceName and PropertyName support wildcard calls.

Security

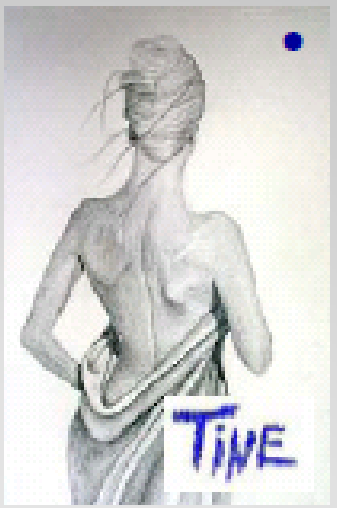
Security

- TINE security based on
  - user name
    - those 16 bytes in the PktHdr
    - to do: use API instead of USERNAME env.
    - done in 4.2.3: allow groups
      - e.g. server can allow all members of ‘mhfe_user’
  - network address
    - from the ethernet packet
    - single address or range
- 3 Levels (cumulative)
  - server
  - property
  - Device
- Access Locks
  - Only the client with the token is allowed access
- Exclusive Read
  - A property can register XREAD in its access parameter
  - XREAD and READ together require an Access lock to be in effect.

Security

- Assigning the ACL information
  - via API
    - e.g. AssignDeviceAccessList()
  - via database configuration file
    - fec.xml (not yet!)
      - => Stock properties to ADD/REMOVE ACL items update the .csv files!
    - e.g. ‘users.csv’, <deviceName>-ipnets.csv, etc.
      - Trying to minimize ‘scanning for files’ at startup by first checking directory for ‘*-ipnets.csv’, ‘*-users.csv’
  - other/better solutions?

DOOCS



DOOCS compatibility

DOOCS compatibility

- Issues
  - most pure acquisition features are mapped !
    - data type mapping is 99%
      - exotic history data types (in progress)
      - TINE struct and bitfield not supported in doocs
        - how much of a problem is that ?
    - full function mapping still an issue
      - e.g. calling P.HIST on a doocs server over a time range, asking for a single int32 value will fail
        - TINE returns the number of points in the interval
      - many such 'gotcha's, but mostly at this (2nd tier) level
    - security (a persistent bother)
      - doocs server must supply the gid/uid of the 'resolved' user seen in the TINE PktHdr.
        - A FEC middle layer will supply the FEC name (definitely not resolvable).
        - Solution: FEC call to a doocs server can set the doocs user to the logged in user (who is hopefully resolvable).

- Issues (continued)
  - 'hidden' stock properties in TINE
    - very easy to 'unhide' at the browser (e.g. rpc_test): just show them.
  - 'hidden' meta-properties in TINE
    - A bit trickier to 'unhide' only the 'relevant' ones
      - acquire full property query information
      - e.g. if 'prpHistoryDepthShort' > 0 then show <property>.HIST in the browser.
      - e.g. if max or min != 0 or units != "" then show <property>.EGU in the browser.
      - etc.
  - some doocs 'favorites' could be added to the meta-property soup:
    - .SYS_MASK will appear in 4.2.3
    - what else ?
  - property servers
    - browse differently !
    - trap the 'has_query_function' status with a call to DEVICES
    - Fill in 'locations' with the results of <property>.NAM at each change of property.

DOOCS compatibility

- Issues (continued)
  - configuration
    - server administrator must remember to set the SVR.GROUP if server is a member of a group
      - e.g. group server BLM consists of 3 servers BLM.1, BLM.2, BLM.3 running on different hosts.
    - should take time to set SVR.TINEFEC
      - provide a 'sensible' FEC name (e.g. "PEVACFEC") to avoid the automatic name of e.g. "Io83a997ab.1f8"
    - make use of SVR.TINEPREF and/or SVR.TINESUFF to decorate a device server name to avoid collisions or ambiguity
      - e.g. SVR.TINEPREF "LASER." would register a server "LASER.ADCSCOPE" instead of "ADCSCOPE"
  - subsystem decorated contexts
    - PETRA.VAC without subsystem leads to context "PETRA" and the server belongs to subsystem "VAC".
    - address resolution does not care:
      - e.g. /PETRA.VAC/IonPump and /PETRA/IonPump both resolve to same server
    - could lead to name collisions in TINE (unless e.g. SVR.TINEPREFF was used)
    - supply a subsystem => the decoration will not be removed
      - but then we end up with a slough of contexts which nominally belong to the same facility.

DOOCS compatibility

- Issues (continued)
  - nice to have:
    - recognize and register MCA properties.
    - Fill in the ‘system stamp’ and/or ‘user stamp’ with e.g. pulse number

DOOCS compatibility

Platforms and APIs

Platforms and APIs

- Supported Platforms
  - Any reason to continue supporting DOS, Win16?
    - if release 3.xx is still supported, they will work
  - VMS may or may not still work
  - Anything else needed?
    - RTEMS ?
    - android ?
  - *Embedded* issues ?
    - is there a disk ?

Platforms and APIs

- Language support
  - C, C++, C# (and .NET), Java
    - native libraries: C and Java
    - everything else interops with the C library
    - C-Lib can be single threaded (tine.dll, libtine.so) or multi-threaded (tinemt.dll, libtinemt.so)
  - Delphi (Lazarus)
    - based on C Lib
    - visual pascal
  - LabView
    - based on C Lib
  - MatLab
    - official 'mex' routines based on C Lib
    - could also use the java Lib
    - octave ?
    - experiences ?
  - Python
    - PyQt, IPython ?
  - Perl ?
  - Functional languages?
    - Scala, F#

- API primarily based on the idea of
  - a *Contract*
    - the requested action/information from the target
  - a *Link*
    - connects the results of the action to the process data
    - specifies a transport mode
      - SINGLE (asynchronous or synchronous)
      - TIMER (POLL)
      - DATACHANGE (REFRESH)
      - EVENT
      - RECEIVE

Platforms and APIs

Platforms and APIs

- APIs
  - C and Java APIs are well known
    - cardinal rule: *don't break the API!*
    - C API is *NOT* object oriented
      - suffers a bit from lack of 'overloading'
        - extended routines:
          - e.g. RegisterDeviceEx(), AttachLinkEx2()
    - Java IS and makes use of a *Link Object* with data acquisition methods !
    - *both*: data is always passed by reference
      - => in Java a scalar is an array of 1 (MatLab does this too!)
    - what is missing, wrong, useless ?
  - '*Official*' C++ API ?
    - (currently there are several)

Platforms and APIs

- APIs (continued)
  - C# and .NET interop with the C Lib but model the API on Java.
    - *except*: everything (even primitives) really is an object and you can pass by reference !
    - structures are easiest in .NET
  - note: with the 'interop' there must be a platform specific library 'tinemt.dll' or 'libtinemt.so' on the path !
    - then can compile with 'anyCPU'
  - ACOP
    - graphics API designed for control
    - originally a common transport API
      - ACOP ActiveX support(ed)
        - TINE, CA, MKI, CDI, ISOLDE, ConSys, etc.
      - acopbeans supports only TINE (and simulation)
        - but with a bit of refactoring ?
          - Interest at KEK to get/set STARS via acopbeans.
      - ACOP.NET is in prototype

- ezTine API ?
  - model on buffered API ?

Platforms and APIs

Platforms and APIs

- Web Tools
  - Web2C ?
  - PhP ?
  - .NET, silverlight ?
  - browser plugin ?
    - instead of http://something.desy.de
    - tine://context/server/device/property

- Command Line tools
  - frequently used in scripts
  - can become *problematic*:
    - each *tget* needs to resolve an address
      - contacts the **ENS** to get the address
      - makes the synchronous call to get the data it wants
      - then exits and forgets everything
  - a forgotten solution:
    - a local repeater runs in the background on the local host
    - 'tget' first checks for a repeater
      - exists:
        - get data from repeater
        - repeater caches the target address and maintains a static listener
      - doesn't exist:
        - do it the brute force way

Platforms and APIs

Central Services

Central Services

- Some have direct relevance to TINE Lib
  - e.g. a starting server *clears* its alarms
    - if the call to "/<myContext>/CAS/RemoveAlarms" is successful -> Yes, the **CAS** is monitoring me !
  - TINE time synchronization expects "/SITE/TIMESRV" to exist
    - if not: no TINE time synchronization
  - if "/<myContext>/Cycler" exists apply the incoming cycle number global to my 'system stamp'.
  - redirect any "<property>.ARCH" call to "/<myContext>/HISTORY"
  - etc.

- Any issues with :
  - naming (ENS/GENS) ?
  - archive system (ARCHIVER/HISTORY) ?
  - post mortem/event (EVENTS) system ?
  - globals system (GLOBALS) ?
  - alarm (CAS/ALMSTATE) system ?
  - state system (STATE) ?
  - statistics system (FECSTATS) ?
  - central logging system (CLOG) ?
  - spy system (CSSPY) ?
- viewing tools, GUIs ?
- specific APIs
  - how do I get this/that from the
    - ENS ?
    - CAS ?
    - etc.

Central Services

Standard Servers

- Standard semi-off the shelf servers
  - motor server
  - scope server
  - video server
  - any other ‘off the shelf' servers ?
  - scan server ?
  - sequencer
  - FSM ?
  - USC (universal slow control)

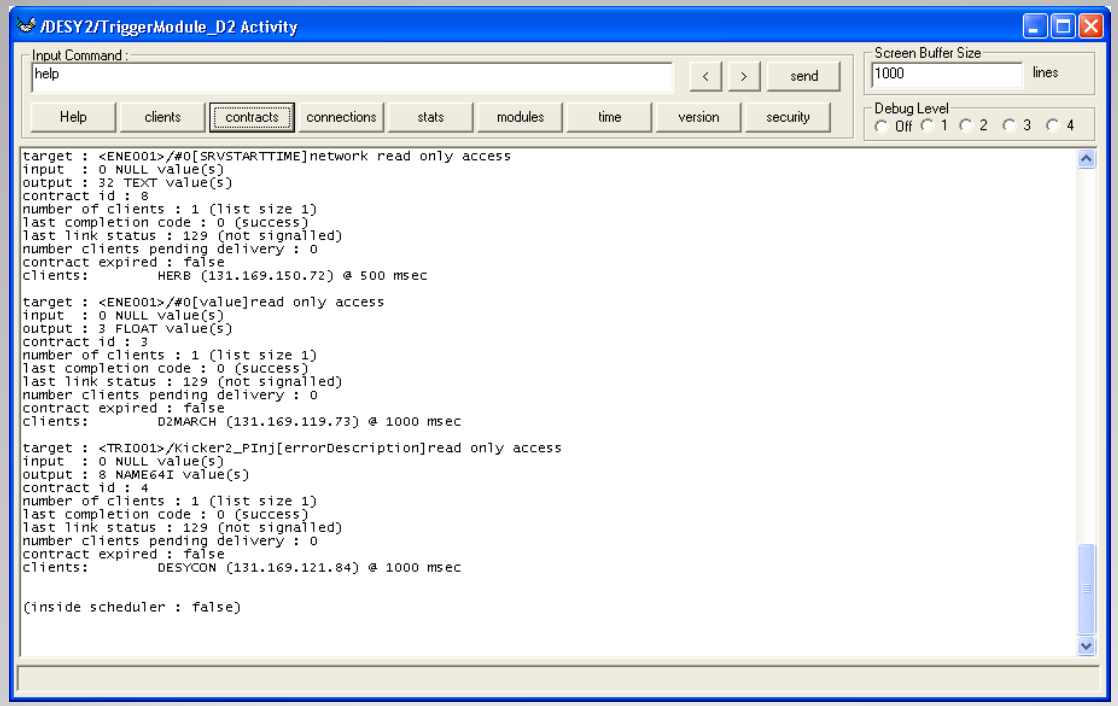
  - tine repeater

- Client-side C library with codecs and other tools?
- Any other issues?

Video System

Peripheral Applications

- watchdogs
  - win32: wdog
  - Linux : autoproc
  - what should they be able to do?
- remote restart daemons
  - wdog, autoproc can do this
  - VxWorks restart task
- application managers ?



diagnostics and logging

diagnostics and logging

- tracing problems …
  - general setup (“nothing works”) problems
    - TINE setup checker (in progress)
      - dump relevant environment variables
      - check connectivity to ENS
      - check manifest
      - check firewall settings
      - etc.
  - Log files
    - location given by FEC_LOG
    - C-Lib:
      - fec.log (1 rotation into fec.bak)
      - LF-CR as per OS
      - format suggestions ?
        - <time> [fec name] (log entry)
        - time zone in <time> constrained to 3 char => standard length
    - Java:
      - Uses java.util.logging.FileHandler
      - LF only
      - fec.log.0 (rotates into fec.log.x)
      - time zone as per JVM locale (e.g. “CET” and “CEST”) => non-standard length
    - all WRITE commands logged by default
    - comments/ suggestions ?

- tracing problems …
  - attachfec
    - normally uses a local PIPE into the FEC process
      - use the FEC name as the PIPE name
    - allows remote access to FEC
    - can also attach to a local client process
      - use the pid as the PIPE name
  - comments/suggestions ?

diagnostics and logging

- CDI
  - active as well as passive CDI servers ?
  - a CDI is a ‘property server’
    - need a ‘device server view’?
  - any issues ?
- TICOM
  - any issues ?

Low Level interfaces

Documentation and Forums

- basic web site (http://tine.desy.de)
  - straight-up doxygen generated
  - other look and feel ?
  - organizational issues ?
  - tutorials ?
  - application videos ?
- mantis (http://tinetracker.desy.de)
- phpbb (http://tineforum.desy.de)
- wiki ?

- .zip and .tar files
- .deb, .rpm, .msi ?
- 'setup' scripts ?
- SVN accessibility ?

Distribution and Repositories

- Where do we go from here ?

Tentative Conclusions

Request header: PktHdr new fields

- HeaderSize
- pid
- Endianness
- Character encoding flag ? (probably not necessary)
- Application ‘string’ (maybe 64 bytes)
  - a short ‘tag’:
    - A middle layer : “FEC”
    - A ‘wrapped’ application: e.g. “MatLab”, “Python”, “Web2C”, “LabView”, etc.
  - + process name
- Reserved fields (not necessary if HeaderSize is in header)

Response Header

- Also needs a PktHdr: We forgot about this!
- Those initial 2 bytes (totalSizeInBytes as UINT16) should become a response header with:
  - totalSizeInBytes
  - HeaderSize
  - endianness
  - FEC name

- sizeInBytes, sizeInElements on request and response
- settable mtu on request side
- unsigned integer format definitions
  - CF_UINT8, CF_UINT16, CF_UINT32, etc.
- display AND setpoint max/min settings ?
  - NO: one set of max, min
  - Can be used for setpoints via call to AssertRangeValid() if developer wants
- return code
  - categorize which return codes an EQM is allowed to use
  - structures
    - status, return code, return source at server, etc.
  - => don't break current API !
  - Java: optional unchecked exceptions ?
    - If ‘some boolean flag’ = true then e.g. throw tineIoException() ?

Other items ...

- Is ‘best source’ !
- Could do load-balancing this way:
  - Instead of “master/slave” use “primary/secondary”
  - primary monitors secondary's NCLIENTS
  - primary needs to redirect to secondary if
    - my NCLIENTS >> his NCLIENTS

software failover

Port Offset

- API to GetMyPortOffset(FECNAME)
  - Check local manifest
    - Found FECNAME -> return assigned port
    - Not found -> return ‘next free port’
  - Could also check with the ENS ?
  - Note the /var/tmp area on Unix is not a good spot for the manifest.
    - Try env variable
    - Then try /var/tine/cache directly
    - Then resort to /var/tmp/tine/cache
    - Or service daemon ?

Meta Properties

- Stock property to return *useable* Meta-Properties
  - “METAPROPERTIES” ?
  - “FILLEDMETAPROPERTIES” ?
  - Also use in Instant Client
    - (show available meta properties check box)

API for dummies and profis

- EZTINE
  - Based on ‘buffered API’ ?
  - Small < 2 pizza) committee to agree on a reduced set of simple API calls (+ tutorial)
- C++
  - Small (< 2 pizza) committee to agree on a C++ API
  - Special aside: use UNASSIGNED_CALLBACK as callbackId in AttachLinkEx() or AttachLinkEx2() to receive the link Id.
- Java
  - Remove ‘final’ from TLink object, etc.
  - New API calls that throw checked exceptions ?
    - (Ahhh, now that’s java!)
  - vs. optional unchecked exceptions ?
    - (violates ‘official’ java style)

Other stuff

- MatLab API
  - Java or C++ ?
- Command line tools (especially tget):
  - Make use of (old) local tineRepeater daemon