SRON HIFI	TEI specification	**Hifi no.: SRON-G/HIFI/SP/2001-001.** **Inst.no.:** n **Issue: Draft 0.1** **Date: 24 April 2001** **Category: -**

Title **TEI specification**

Prepared by : Johan Wezelman Date :

Albrecht de Jonge

Checked by : Date :

Agreed by : Date :

Authorised by : Date :

Filename: C:\My Documents\Johan\documentatie TEI\TEI specification V0.1.doc

Distribution

Document Change Record

Issue	Date	Changed Section	Description of Change

Table of contents

1 INTRODUCTION

The HIFI Instrument Level tests are conducted by the SCOS 2000 via TM/TC packets.
Test equipment is linked to the TM/TC interface via Test-Equipment Interfaces (TEI's).
This document describes the specification of the TEI's.

2 DESIGN OVERVIEW

2.1 General

Many different types of interfaces are required because of the large variety of test equipment. The TEI described in this document is designed for maximum flexibility and should be able to provide communication with every specific piece of test equipment needed for the tests.

2.2 Hardware

The TEI hardware can roughly be compared with a standard PC without keyboard and monitor. To reduce its size the TEI is based on PC104 Standard boards. To provide the TEI with the required flexibility it can be equiped with a variety of different interfaces: Serial, Parallel, IEEE488, Analog & Digital inputs etc..
An ethernet interface connects the TEI to the TM/TC interface.

2.3 Programming language

Programs for the TEI prototype were written in C and gcc was used as compiler and linker. (also see: 2.5 Portability issues)
As an alternative C++ can be used.

2.4 Operating system

The TEI's use a minimized Linux system based on the Red-Hat 6.2 distribution and RT Linux. We use Linux version 2.2.18-rtl.
Since the Operating System together with the Application Programs should fit in the 32Mb Flash-Disk on the CPU board, a Kernel with a minimum set of utilities is used.

2.5 Portability issues

The prototype TEI uses RT Linux (Linux version 2.2.18-rtl), C programs and gcc as a compiler and linker (gcc version egcs 2.91.66).
If a compatible alternative is used for the development environment, then all or part of the already developed system components can be integrated.
Below the compatibility of the Separate System Components are discussed:

2.5.1 Router communication

(a) *The TCP/IP Interface is Linux specific*
(b) *The EGSE client and PSICD libraries:*
TBS
(c) *The EGSE client and PSICD source code.*
The TCP/IP Interface is Linux specific
Non-ANSI C standard properties of the gcc C language are used, so a gcc compatible compiler is required.

2.5.2 Application integration

(a) *The skeleton program source code:*
An ANSI C compiler supporting POSIX threads is required.

2.5.3 Application development

(a) *Libraries for various equipment interfaces*
TBS

(b) *Source code for various equipment interfaces.*
Linux specific libraries are used to communicate with serial and parallel ports.

(c) *Real time modules*
Real time Linux specific modules are used.

3 HARDWARE

3.1 General concept

The TEI hardware is based on PC104 Standard boards purchased from different companies. The TEI can contain up to three different PC 104 boards. One of these boards is the CPU board which provides the TEI box with the following features:

- 1 Ethernet Connection (see: 3.3)
- 4 Serial Ports (see: 3.3)
- 1 Parallel Port (see: 3.3)

Depending on the requirements for a specific interface, extra features can be selected by choosing the appropriate boards for the other two PC104 slots. The possibilities include:

- 16 Analog inputs (see: 3.4)
- 4 Analog outputs (see: 3.4)
- 8 digital outputs (see: 3.4)
- 8 digital inputs (see: 3.4)
- an IEEE 488 Port (see: 3.5)
- a Camera interface (see:3.6)
- a Quadrature decoder (see:3.7)

3.2 Box and power supply

Three of the modules in which the TEI's are housed fit in one 19" Rack.

The standard TEI box contains:

(a) A single 5V power supply.
(b) a PC104 Standard CPU board.
(c) a PC104 Standard A/D interface board.
(d) a PC104 Standard IEEE488 interface board.

As an alternative the boards (c) and/or (d) can be replaced with for example:

(e) a PC104 Standard Camera interface.
(f) a PC104 Standard Quadrature Decoder.

3.3 CPU board

The CPU board used is the CPU-1220 produced by Eurotech (http://www.eurotech.it)
This board contains an almost complete standard PC.
Main Features:

cpu	66 MHz, 486DX2 equivalent
memory	32 Mb ram
Solid state disk	32 Mb M-systems Disk-On-Chip
Serial interface	4 ports, 16550 compatible, COM3 and COM4 RS232 only, COM1 and COM2 RS232/485/422 selectable
Parallel interface	IEEE 1284, supports EPP/ECP/SPP
Floppy disk drive	As alternative to parallel port, BIOS configured
Keyboard, monitor, mouse i/f	PC keyboard, SVGA, PS/2 mouse
Ethernet	10/100Mb Twisted pair

3.4 A/D interface board

The A/D interface board used is the Diamond-MM-16-AT produced by Diamond Systems (http://www.diamondsystems.com).
It provides both Analog and Digital inputs and outputs.
Main features:

Analog inputs:	16 analog inputs, 16-bit resolution 100KHz max sampling rate Programmable gain and range (0..+V or +/-V ; range = 10, 5, 2.5 or 1.25V) FIFO for gap-free sampling (512 samples)
Analog outputs:	4 12-bit analog outputs Programmable analog output range (0..+V or +/-V ; range 0..10V in 1mV steps programmable) Autocalibration of both A/D and D/A
Digital inputs:	8 digital inputs, TTL compatible
Digital outputs:	8 digital outputs, TTL compatible

3.5 IEEE488 interface board

The IEEE 488.2 (GPIB) interface board used is produced by Computer Boards. (http://www.computerboards.com)
Main features:

Data buffer	1024 word fifo
Data Transfer Rate:	Up to over 1Mbytes/second
Interface	CB7210.2 GPIB chip

3.6 Camera Interface

TBS.

3.7 Quadrature Decoder Interface

TBS.

4 OPERATING SYSTEM

4.1 Kernel configuration and build

TBS

4.2 Minimum installation content

TBS

4.3 OS support: shell, ftp, tar, …

TBS

5 APPLICATION PROGRAMS

5.1 EGSE interface library

To be written...

5.2 Generic structure

Typical characteristics of the Application Software:
- Multi-threaded programs.
- Time critical processes are handled in Real-Time modules.
- The Real-Time modules communicate via FIFO buffers with normal processes.

In every TEI the system comprises three different types of threads:
- Event Threads
- Measure Threads
- Command Threads

An Event Thread is waiting for its type of events to happen and sends Event Reports to the Router. A Measure Thread periodically sends its housekeeping reports to the Router.
A Command Thread is waiting for its specific Commands and initiates the execution.

5.3 Commands and telemetry Interface

The interface to support a TEI application in the exchange of TM and TC packets via the Herschel router with SCOS2000

5.3.1 Packet definition

The type `Packet` is defined to support the exchange of TC/TM packets:

```
typedef struct { word service ;  byte * content ;  int length ; } Packet ;
```

`service`	The service type and subtype of the packet encoded into one 16-bit word. The encoding is: most significant 8 bits = type, least significant 8 bits = subtype.
`content`	A pointer to a byte array containing the 'source data' field of the TNM packet or the 'application data' of a TM packet; without 'packet header', 'data field header', and 'packet error control' fields. The byte array should be large enough to contain the data.
`length`	The number of bytes actually used of the 'content' array

5.3.2 Services

Macro definition:

```
SERVICE(type,subtype)
```

This macro is provided to fill the 'service' field in the Packet structure with a proper type/subtype combination.

A number of predefined service definitions is available:

`TELECOMMAND_ACCEPTANCE_REPORT_SUCCES`	`SERVICE(1,1)`
`TELECOMMAND_ACCEPTANCE_REPORT_FAILURE`	`SERVICE(1,2)`
`HOUSEKEEPING_PARAMETER_REPORT`	`SERVICE(3,25)`
`EVENT_REPORT`	`SERVICE(5,1)`
`START_FUNCTION`	`SERVICE(8,1)`
`STOP_FUNCTION`	`SERVICE(8,2)`
`PERFORM_ACTIVITY_FUNCTION`	`SERVICE(8,3)`
`LOAD_PARAMETERS_FUNCTION`	`SERVICE(8,4)`
`PERFORM_CONNECTION_TEST`	`SERVICE(17,1)`

5.3.3 Parameter value handling

Three macros are defined to support the handling of non-byte parameters encoded into a byte array.

5.3.3.1 Macro DEF_PAR

```
#define DEF_PAR(name, offset, shift, length )
```

This macro defines the properties of a parameter.

`name`	A unique name of the parameter. The macro will generate some symbols beginning with `name_`.
`offset`	The offset (in bytes) into the byte array where the parameter is located. Normally even if parameters are word aligned.
`shift`	The number of bytes to shift the parameter with respect to word alignment. Zero means that the least significant bit of the parameter is the least significant bit of a 2 byte word.
`length`	The number of bits in the parameter.

5.3.3.2 Macro SETPAR

```
#define SETPAR(buf, name, val) \
```

This macro inserts a parameter into a byte array.

`buf`	The byte array where the parameter must be inserted
`name`	The unique name of the parameter, as given earlier in a DEF_PAR macro.
`val`	The value for the parameter to be inserted.

5.3.3.3 Macro GETPAR

```
#define GETPAR(buf,name)
```

This macro retrieves a parameter from a byte array.

`buf`	The byte array from where the parameter must be retrieved
`name`	The unique name of the parameter, as given earlier in a DEF_PAR macro.
`(value)`	The value of the macro is the value of the retrieved parameter.

5.3.3.4 Example:

Take the following code fragment:

```
DEF_PAR ( ZZZ, 6, 4, 8 )
for ( i = 0 ; i < 10 ; i ++ ) b[i] = i + ( i << 4 );
for ( i = 0 ; i < 10 ; i ++ ) printf ( "%4.4x", b[i] );
printf ( "\n%4.4x\n", GETPAR( b, ZZZ ));
SETPAR ( b, ZZZ, 0xAA );
for ( i = 0 ; i < 10 ; i ++ ) printf ( "%4.4x", b[i] );
```

This describes the 8-bit parameter ZZZ shifted 4 bits in the 2-byte word starting at byte 6. The code will generate the output:

```
  0   11   22   33   44   55   66   77   88   99
 67
  0   11   22   33   44   55   6A   A7   88   99
```

5.3.4 Packet I/O functions

5.3.4.1 Function startTMTC

```
int startTMTC ( int apid, int EGSEClientFd );
```

`apid`	The application program ID to be used in the packet headers
`EGSEClientFD`	The file descriptor obtained from the EGSE router interface.
`(value)`	zero for a succesfull call.

This function initializes the packet structure interface. Only a single apid value per application/router connection is supported. The function will tell the router to forward TC packets intended for this apid to this application.

5.3.4.2 Function closeTMTC

```
void closeTMTC();
```

This function unregisters the current apid with the router. The connection to the router itself is not broken.

5.3.4.3 Function sendTM

```
int sendTM ( Packet * p );
```

This function will send a type `p->service` TM packet containing `p->length` bytes of source data from `p->content`. The packet header, data field header and packet error control fields of the TM packet are provided by this function.

5.3.4.4 Function sendTMtime

```
int sendTMtime ( Packet * p , TimeTag tt );
```

This function is identical to `sendTM`, except that the time tag of the data field header is filled from the `tt` parameter, instead of being set to the time of function call.

5.3.4.5 Function getTCwait

```
int getTCwait ( Packet * p );
```

This function will fill set the pointer `p->content` to a static area containing the bytes of the application data field of the next TC packet for this APID. It will set `p->length` to the number of valid bytes in the application data field. The function will set
`p->service` to the service type/subtype of the TC packet. The packet error control field of the packet has been verified. The packet service type/subtype has been checked to match the list given in the last `setTCservices` function call. The function blocks until a valid, complete packet has been received.
As soon as possible after a successful call to `getTCwait` and, and before the next call to `getTCwait` or `getTC`, the caller must call one of the functions `acceptTC` or `rejectTC`.

5.3.4.6 Function getTC

```
int getTC( Packet * p );
```

This function operates identically to `getTCwait` except that it will not block but immediately return with value 0 if no complete valid packet was received at the moment of the call.

5.3.4.7 Function acceptTC

```
void acceptTC() ;
```

This function will send a `TELECOMMAND_ACCEPTANCE_REPORT_SUCCES` packet.

5.3.4.8 Function rejectTC

```
void rejectTC() ;
```

This function will send a `TELECOMMAND_ACCEPTANCE_REPORT_FAILURE` packet.

5.3.4.9 Function setServices

```
void setTCservices ( int * services, int count );
```

This function sets the service type/subtypes that will be considered valid for TC packets to the `count` values given in the array `services`. Any new call to this function will supersede the previous setting. The service (17,1) `PERFORM_CONNECTION_TEST` is always handled by the software internally

5.4 Example code

5.4.1 main program

```
#include "EGSEclient.h"
#include "PSICD.h"

#include <stdio.h>
#include <pthread.h>

void* measureTaker(void * arg );
void* commandWatcher( void * arg );
void* eventWatcher( void * arg );

int main( int argc , char ** argv )
{       int result ;
        pthread_t eventThread, measureThread, commandThread ;

        if ( argc != 5 ) {
                fprintf ( stderr, "Usage: %s router port myname apid\n",
                                                argv[0] );
                return 1 ;
        }
        startTMTC (
                atoi(argv[4]) ,
                EGSEClientOpen ( argv[1], atoi(argv[2]), argv[3] )
                );
        result = pthread_create ( & eventThread, NULL, eventWatcher, NULL );
        result = pthread_create ( & measureThread, NULL, measureTaker, NULL );
        result = pthread_create ( & commandThread, NULL, commandWatcher, NULL );
        /* wait for threads to die, ignoring their returns */
        pthread_join ( eventThread, 0 );
        pthread_join ( measureThread, 0 );
        pthread_join ( commandThread, 0 );
        closeTMTC();
        return 0 ;
}
```

5.4.2 command watching thread

```
void* commandWatcher( void * arg ){
        Packet command ;
        while (1 ) {
                getTCwait( & command );
                switch ( command.service ) {
                case START_FUNCTION:
                        /* ... */
                        acceptTC();
                        break ;
                case STOP_FUNCTION:
                        /* ... */
                        acceptTC();
                        break ;
                case PERFORM_ACTIVITY_FUNCTION:
                        /* ... */
                        acceptTC() ;
                        break ;
                default:
                        rejectTC() ;
                }
        }
}
```

5.4.3 measuring thread

```
void* measureTaker(void * arg ){
        byte measured[24] ;    /* to contain telemetry source data */
        Packet p ;
        p.service = HOUSEKEEPING_PARAMETER_REPORT ;
        p.content = measured ;
        p.length = sizeof measured ;
        while(1) {
                usleep(1000000);        /* or other wait until ... */
                /* collect measurements into array 'measured' */
                sendTM ( &p, 0 );
        }
}
```

5.4.4 event handling thread

```
void* eventWatcher( void * arg ){
        byte eventData[12] ;
        Packet p ;
        TimeTag tt ;
        p.service = EVENT_REPORT ;
        p.content = eventData ;
        p.length = sizeof eventData ;
        while(1) {
                usleep(1000000);
                /* read event - must block until event happened and
                        read is done */
                /* put event data into 'eventData' and event time into 'tt' */
                sendTM ( &p, tt );      /*  use 'tt' as packet time tag */
        }
}
```

6 DEVELOPMENT

6.1 Development box

For the development of the TEI software we used a 486 desktop computer.
For the operating system we used: RT Linux (Linux version 2.2.18-rtl)

6.2 Development utilities

The TEI prototype was built on a RT Linux system (Linux version 2.2.18-rtl) and we used gcc to compile and link the C source files (gcc version egcs 2.91.66).
Other tools and utilities from the Red-Hat Linux 6.2 distribution were also available.

6.3 Test rig

The test rig is a set-up in which the TEI CPU board can be connected with a diskdrive, keyboard, monitor and network connection. It is needed for the initial installation of the software and for test-purposes.