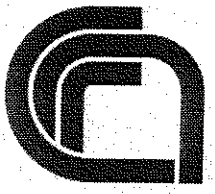
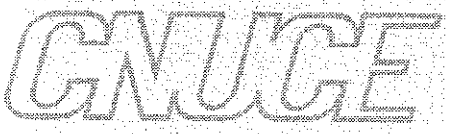


Consiglio Nazionale delle Ricerche

HS - WinLib Descrizione delle librerie e relativo protocollo di comunicazione comunicazione PC ⇔ HS-MAX

R.Bianchi Bandinelli e A.Guerra

CNUCE: C 95 - 35



HS-WinLib

Descrizione delle librerie
&
relativo protocollo di comunicazione
PC ⇔ HS-MAX

CNUCE *©TRIALOG*

Rapporto interno C95-35

Ottobre 1995

Rolando Bianchi Bandinelli

Andrea Guerra

Generalità

Le librerie HS_WinLib della TRIALOG consentono di scrivere dei programmi in C in ambiente MS-WINDOWS per realizzare dei device EHS su mezzi trasmissivi PL (Power Line) o TP1 (Twisted Pair 1). Lo sviluppo completo di un device EHS con tali librerie è legato all'utilizzo delle schede HS_MAX prodotte dalla stessa Società.

Requisiti di sistema

- PC MS-DOS compatibile (ver. 3.00 o superiori);
- Windows 3.0 o 3.1;
- Ambiente di sviluppo C++ Microsoft (v.7 o v.8) o Borland (v.3 o v.4).

Configurazioni EHS supportate

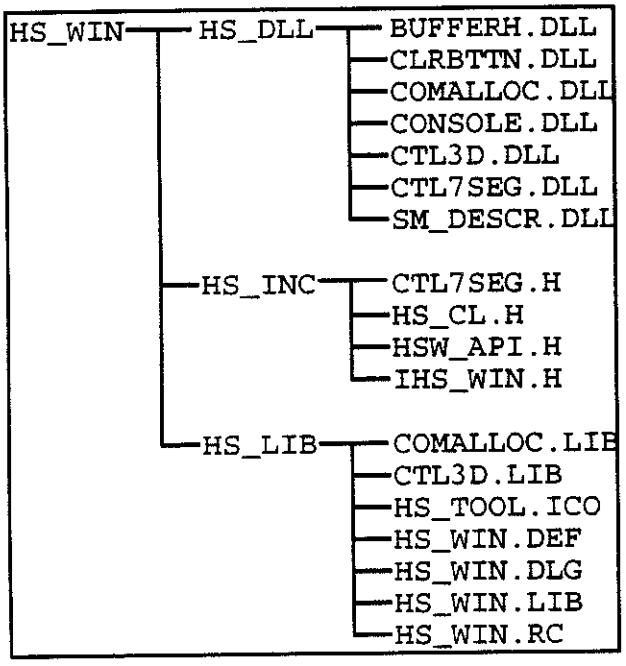
Le HS_WinLib consentono di costruire le seguenti unità:

- FC (Feature Controller);
- CoD (Complex Device);
- FC e CoD.

Descrizione del pacchetto sw

Le HS_WinLib nel PC sono la controparte delle HS-Lib e dell'HSTracer, librerie EHS prodotte dalla TRIALOG fornite per sviluppare SW su microcontrollore. Le due interfacce di pregrammazione sono molto simili, sebbene le HS_WinLib siano più semplici da utilizzare, dal momento che non sono limitate dall'uso del microcontrollore.
Incluso nelle HS_WinLib è anche un sottoinsieme di watch (funzioni di supervisione) per l'HSTracer a scopo di debugging.

Il sistema di librerie HS_WinLib consiste in un insieme di file suddivisi in 3 subdirectory:



In particolare, i file header nella directory **HS_INC** contengono le seguenti informazioni:

- **HSW_API.H**: definizione di tipi, costanti, macro, variabili, prototipi di funzioni da utilizzare nel codice dell'applicazione;
- **HS_CL.H**: definizioni dei valori delle costanti del linguaggio dei comandi usate negli esempi dati;
- **IHS_WIN.H**: file necessario alla compilazione dei file risorse Windows.

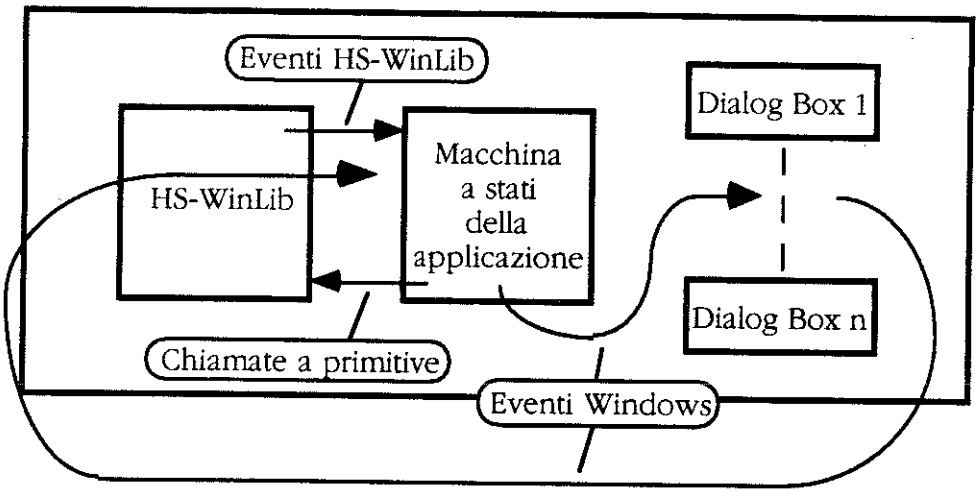
I file in codice sorgente C devono includere i file header nell'ordine in cui sono stati elencati.

Le operazioni da seguire nella realizzazione di un device sono le seguenti:

- Compilazione del file **APPLI.C** (sorgente dell'applicazione), ottenendo il file **APPLI.OBJ**;
- Compilazione dei file **HS_WIN.RC**, **HS_WIN.DLG**, **APPLI.DLG** (file che contiene la descrizione delle dialog box dell'applicazione), ottenendo il file **WIN_API.RES**;
- Linking dei file ottenuti dai precedenti passi con i file **HS_WIN.LIB** e **HS_WIN.DEF**.

Filosofia di funzionamento di un'applicazione HS_WinLib

Un'applicazione HS_WinLib si basa sul concetto di *Macchina a Stati*, ovvero una procedura la cui esecuzione è basata sugli eventi generati dalle HS_WinLib.



Tale procedura, dal nome predefinito **ApplicationSM()**, deve risiedere nel file sorgente dell'applicazione (`APPLI.C`).
Gli eventi all'occorrenza dei quali le HS_WinLib chiamano la procedura `ApplicationSM()`, sono di due categorie:

Windows event

- **cInitEvent**: questo evento occorre quando è avviata l'applicazione Windows. Esso consente all'applicazione di avviare attività di inizializzazione riguardanti la gestione delle finestre (es.: creazione di una finestra), la configurazione di device EHS (es.: settaggio dello unique code mediante la primitiva **SetUniqueCode** o leggendo un file di configurazione utilizzando la primitiva **ReadUnitConf**) e le attività iniziali di gestione della rete (es.: avviamento della registrazione tramite la primitiva **StartRegistration**).

- **cWindowsEvent**: l'occorrenza di questo evento ha luogo per la gestione di un messaggio Windows, ma che non viene trattato dalle HS-WinLib. La macchina a stati è chiamata anche se tale evento non è atteso.

- **cEndEvent**: evento che occorre quando termina l'esecuzione dell'applicazione Windows. Esso fa sì che la macchina a stati sia chiamata automaticamente e non dovrebbe essere usato come parametro per una chiamata alla funzione **WaitEvent**. Questo evento consente di cancellare ogni risorsa Windows allocata all'occorrenza dell'evento **cInitEvent**.

HS-WinLib event

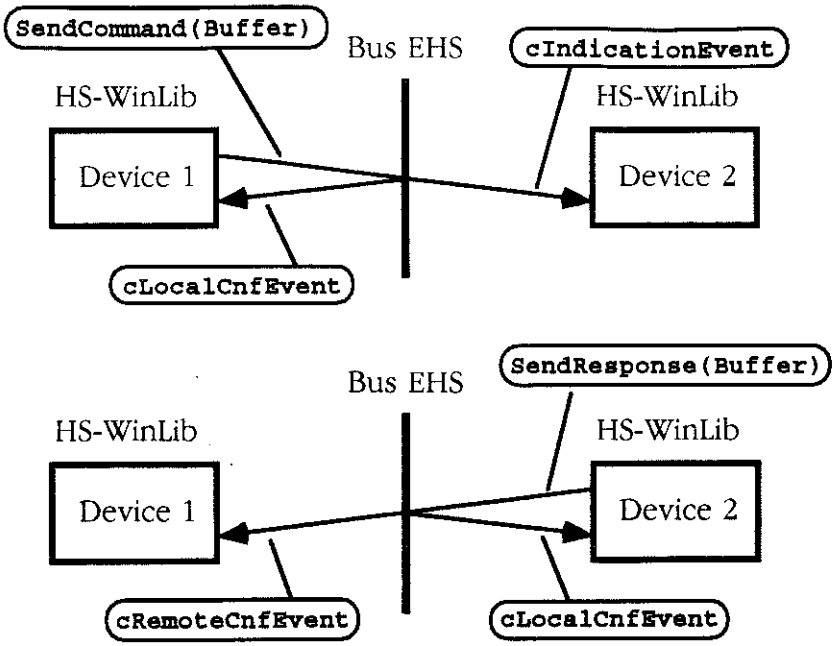
- **cIndicationEvent**: l'occorrenza di questo evento ha luogo al momento della ricezione di un comando.
- **cRemoteCnfEvent**: evento che occorre quando viene ricevuta una risposta dal device remoto.
- **cLocalCnfEvent**: evento occorrente dopo avere trasmesso un comando (funzione **SendCommand**) o una risposta (funzione **SendResponse**).
- **cRegistrationEvent**: questo evento occorre dopo il completamento della fase di registrazione. Se quest'ultima ha esito positivo (scuccesso), l'unità PC ha guadagnato il proprio indirizzo sulla sottorete EHS, dopodiché viene avviata automaticamente la fase di enrolment. Lo stato della registrazione può essere testato leggendo la variabile **RegState**.
- **cEnrolmentEvent**: questo evento occarre dopo il completamento della fase di enrolment. Nel caso di enrolment visto da un FC, tale evento indica che un nuovo CoD è stato enrolled o che un CoD esistente è scomparso. Le informazioni riguardo detta fase di enrolment possono essere tratte dalle variabili **NumEnrolledCoDs** e **vEnrolledCoDs**. Nel caso di enrolment visto da un CoD, tale evento indica che una nuova FC ha enrolled il CoD stesso o che un FC che aveva precedentemente enrolled un CoD è stato sconnesso dalla rte. Le informazioni riguardo detta fase di enrolment possono essere tratte dalle variabili **NumEnrolledFCs** e **vEnrolledFCs**.
- **cTokenPasingEvent**: evento occorrente in un FC quando è stata cambiata una condizione di token passing. Il tipo di cambiamento ed il CoD ad esso associato possono essere monitorizzati nella struttura lobale **ApplicationDialog**.

Trasmissione e ricezione di comandi

Un comando è costituito da un frame inviato sulla rete EHS da un'applicazione ad un altro device. Alcuni campi del frame, non visibili a livello di API, sono gestiti dalle HS-WinLib e non riguardano l'applicazione. Un API definisce una struttura dati chiamata **tEHS_Buffer**. Alcuni campi riguardanti l'applicazione sono riempiti con valori di default dalle HS-WinLib tramite l'utilizzo della funzione **InitTxBuffer**, e sono direttamente modificabili dall'applicazione trasmittente.
Altri campi non hanno valori di default (vedi **Id**) e devono essere riempiti dall'applicazione prima della trasmissione.

La trasmissione di comandi o risposte EHS avviene mediante le primitive **SendCommand**, **SendResponse**, **SendL7Cnf**.

La primitiva **SendCommand**, eseguita dall'applicazione trasmittente, dà luogo all'occorrenza dell'evento **cLocalCnfEvent** nella stessa applicazione (il risultato dell'operazione può essere letto nella variabile **TxResult**) e all'evento **cIndicationEvent** nel device ricevente (il comando ricevuto può essere letto nella variabile globale predefinita **RxBuffer** - struttura **tEHS_Buffer**).
La primitiva **SendResponse**, eseguita dal device che ha rilevato l'evento **cIndicationEvent**, dà luogo all'occorrenza dell'evento **cLocalCnfEvent** nel device stesso (il risultato dell'operazione risiede nella variabile **TxResult**) e all'evento **cRemoteCnfEvent** nel device dell'API che aveva inviato il comando stesso (i dati associati a tale evento possono essere letti nella variabile **RxBuffer**).
La figura seguente schematizza il flusso relativo alle primitive menzionate e ai relativi eventi.



I comandi sono suddivisi in 2 categorie:

- **Comandi automaticamente confermati**: richiedono sempre una risposta dall'applicazione ricevente (es.: Read - comando che richiede un report al device indirizzato).
- **Comandi non automaticamente confermati**: non richiedono automaticamente una risposta dall'applicazione ricevente. Per tali comandi, la conferma deve essere richiesta esplicitamente dall'applicazione trasmittente riempiendo opportunamente il campo **ConfType** del buffer di trasmissione (struttura **tEHS_Buffer**).

La distinzione tra i due tipi di comandidipende dal valore del campo **Service** del buffer di trasmissione (struttura **tEHS_Buffer**).

: Anche nel caso di comandi automaticamente confermati è necessario riempire il campo **ConfType** del buffer di trasmissione per definire il tipo di conferma richiesta (**cSucc** o **cFull**, a seconda che si voglia conferma in caso di solo successo o in ogni caso).

Ricezione di una conferma locale

Dopo avere inviato un comando (**Sendcommand**) o una risposta (**SendResponse**) l'applicazione deve usare la primitiva **WaitEvent** per informare le HS-WinLib che attende una conferma locale. Le HS-WinLib, a loro volta, aggiornano il contenuto della variabile **TxBuffer** e segnalano l'evento (**cLocalCnfEvent**) alla macchina a stati.

Ricezione di una indicazione

La ricezione di un'indicazione da parte di un device ha luogo quando un'applicazione trasmittente invia un comando. Onde evitare che il comando vada perduto, è necessario che la macchina a stati dell'applicazione ricevente preveda nel codice relativo l'evento **cIndicationEvent**. E' obbligatorio attendere regolarmente l'evento **cIndicationEvent**.

Ricezione di una conferma remota

Un comando che richiede una risposta (sia esso con conferma automatica o meno) fa sì che successivamente le HS-WinLib informino l'applicazione che una conferma remota è ricevuta, svegliando la macchina a stati con la generazione dell'evento **cRemoteCnfEvent**. Il risultato dell'operazione può essere letto nel campo **Result** della variabile **RxBuffer** dell'applicazione che ha inviato il comando. Onde evitare che la risposta vada perduta, è necessario che la macchina a stati dell'applicazione che ha inviato il comando preveda nel codice relativo l'evento **cRemoteCnfEvent**. E' obbligatorio attendere regolarmente l'evento **cRemoteCnfEvent**.

Trasmissione di una risposta ad un comando non automaticamente confermato

Per capire se debba essere spedita o meno una risposta all'applicazione trasmittente, può essere letto il campo **ConfType** della variabile **RxBuffer**. Comunque, dato che le HS-WinLib effettuano il controllo sul tipo di conferma, l'applicazione ricevente non invia risposte che non sono richieste. Ciò consente a quest'ultima di non curarsi del campo **ConfType** e inviare una risposta in ogni caso.
Quando una risposta è necessaria, l'applicazione ricevente dovrebbe utilizzare la primitiva **SendL7Cnf** con l'opportuno parametro EHS.

Trasmissione di una risposta ad un comando automaticamente confermato

Questa operazione è analoga a quella relativa al comando non automaticamente confermato. L'unica differenza sta nel fatto che anziché usare la primitiva **SendL7Cnf** deve essere utilizzata la primitiva **SendResponse** e il parametro da passare a quest'ultima è la variabile **RxBuffer**, dopo avere opportunamente modificato i campi **DataLength** e **Data[]**.

Formato corretto dei comandi

Dal momento che un comando o una risposta con formato scorretto non viene spedito, è importante conoscere se il formato stesso sia corretto o meno.
L'applicazione viene informata dalle HS-WinLib sull'eventuale scorrettezza di formato del comando (o risposta) spedito. Ciò avviene tramite l'evento di conferma locale: nel caso in cui il formato non sia corretto, la variabile **TxResult** conterrà il valore **cL7_CantSend**. Di seguito sono riportati i motivi per cui il formato può ritenersi scorretto:

Richiesta (comando)

- L'identificatore del destinatario (**Id**) non è noto;
- Il tipo di conferma (campo **ConfType**) di un comando automaticamente confermato è stato settato al valore **cFail** o **cNone**.

Risposta

L'identificatore del destinatario (**Id**) non è noto;
Il tipo di conferma (campo **ConfType**) è in contraddizione con il valore del risultato (es.: tipo di conferma **cSucc** con valore di errore per risultato).
Il flag **StopOnError** è **TRUE** mentre il valore del risultato corrisponde ad una condizione di errore.

Blocco di comandi

E' possibile impacchettare diversi comandi in un singolo frame. Questa caratteristica consiste nel **meccanismo di Blocking**. Il principio di questo meccanismo è basato sulla trasmissione e ricezione di comandi come visto precedentemente. La differenza sta esclusivamente sul settaggio del flag **Consecutive** (campo della struttura **tEHS_Buffer**), il cui valore di default è **FALSE** (comando unico).

Il flag **Consecutive** uguale a **FALSE**, significa che non c'è alcun blocking (il comando da spedire è unico), pertanto il frame viene trasmesso immediatamente, e una conferma locale segue la chiamata alla primitiva **SendCommand**.

Il flag **Consecutive** uguale a **TRUE**, indica che oltre al presente comando ne deve essere trasmesso ancora un altro nello stesso frame; in tal caso, la trasmissione è ritardata.

In un gruppo di comandi, il flag **Consecutive** di ognuno di essi deve eseere settato al valore **TRUE**, eccetto per l'ultimo comando del frame. Il campo **Id** che specifica il destinatario di un comando deve essere il medesimo per tutti i comandi che sono nello stesso gruppo.

Il meccanismo di blocking è trasparente all'utente. Infatti, nel caso di un frame composto da più comandi l'applicazione ricevente risponde ad ognuno di essi come se fossero stati spediti individualmente (da ciò si deduce che tale meccanismo viene opportunamente codificato a più basso livello dalle schede HS-MAX).
Se si verifica una condizione di errore (ottenuto da conferma remota) nel mezzo della trasmissione di una serie di comandi in uno stesso frame, l'applicazione trasmittente può voler abortire la spedizione dei comandi successivi alla verifica dell'evento. Per effettuare tale operazione è necessario settare a **TRUE** il valore del campo **StopOnError**. Il device remoto dovrebbe utilizzare lo stesso valore di detto campo nel preparare la conferma remota.
Se il campo **Result** della conferma remota indica una condizione di errore (**cBadRoute**, **cDataOutOfRange**) ed il valore del campo **StopOnError** è **TRUE**, la trasmissione del resto dei comandi appartenenti a quel blocco viene cancellata.
L'applicazione non deve testare questi campi di conferma remota dal momento che tale servizio è fornito direttamente dalle HS-WinLib.

Contenuti dei dati

E' essenzialeindicare la lunghezza (espressa in numero di byte) dei dati **Data[]** nel campo **DataLength** del buffer prima della trasmissione.

Comandi di gruppo

Un'applicazione può spedire un comando ad un gruppo di device oltre che ad un device individuale. Gli identificatori predefiniti da 1 a 3 contengono informazioni precodificate per comandi di gruppo. Di seguito sono riportate le specifiche relative a detti identificatori.

- **Id = 1**: è utilizzato per raggiungere il gruppo costituito da tutti i Feature Controller (indirizzo di gruppo 20h). Se un'applicazione vuole trasmettere un comando a tutti i FC, deve preventivamente settare ad 1 il campo **Id** del buffer.

- **Id = 2**: è utilizzato per raggiungere il gruppo costituito da tutti i Complex Device (indirizzo di gruppo C0h). Se un'applicazione vuole trasmettere un comando a tutti i CoD, deve preventivamente settare ad 2 il campo **Id** del buffer.

- **Id = 3**: è utilizzato per raggiungere il gruppo costituito da tuttie le unità della rete (indirizzo di gruppo FFh). Se un'applicazione vuole trasmettere un comando a tutti i FC, deve preventivamente settare ad 1 il campo **Id** del buffer.

Si noti che la trasmissione di un comando di gruppo può generare nella rete una notevole distribuzione di traffico se le risposte sono richieste da tutti i device. E' quindi opportuno cercare di evitare di trasmettere comandi di gruppo che richiedano risposta, dal momento che tutti i device destinatari proveranno a rispondere; ciò, infatti, non fa altro che generare dei conflitti di accesso al bus.
Nella trasmissione di un comandi di gruppo, quindi, è necessario esplicitare che solamente pochi device specifici dovranno rispondere al comando. Per esempio, sel il campo **`ConfType`** del buffer di trasmissione è riempito con il valore **`cSucc`**, una conferma remota dovrà essere inviata solamente da chi ha eseguito con successo il comando stesso.
Un altro modo per limitare le risposte consiste nell'utilizzare il meccanismo di blocking. I primi comandi di un pacchetto devono avere il campo **`StopOnError`** settato al valore **`TRUE`**; questi comandi sono quindi usati come filtri. In tal caso, solamente i device destinatari che riceveranno l'indicazione di detti primi comandi dovranno rispondere.

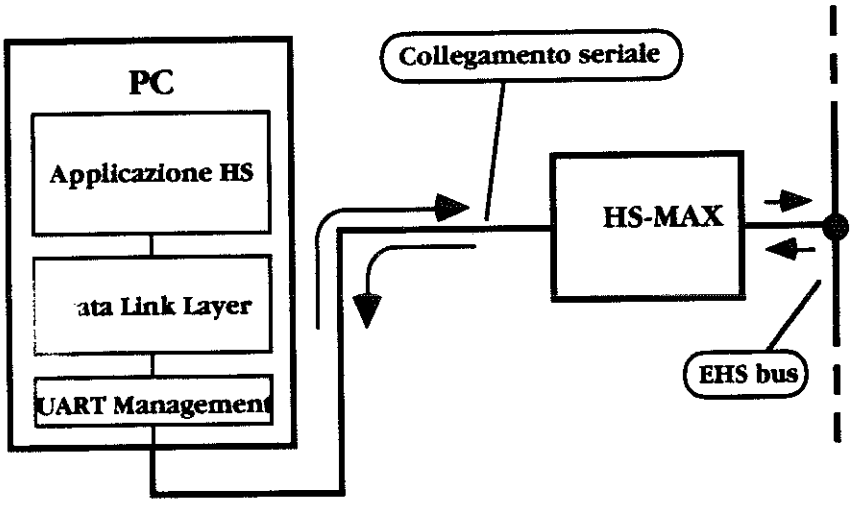
Un comando di gruppo non è confermato a livello L2 (Data Link Layer - MAC sublayer). Perciò, un tale comando non è sicuro del fatto che il ricevitore non spedisca conferma locale e così il trasmettitore non ha modo di sapere se il messaggio sia stato ricevuto.

Strutture dati e variabili definite nelle HS-WinLib

Le strutture dati e le variabili utilizzate nella costruzione di un'applicazione sono definite nei file header precedentemente menzionati.
Si noti che la variabile **`RxBuffer`** di tipo **`tEHS_Buffer`** è predefinita e viene riempita dalle HS-WinLib all'atto della ricezione di un comando, mentre non è predefinita alcuna variabile **`TxBuffer`**, in quanto l'applicazione può non richiederne alcuna (caso in cui sia prevista solamente la ricezione - es.: dimmer), come può richiederne più di una (caso in cui sia previsto trasmettere più comandi diversi).

Gestione delle schede HS-MAX

Le schede HS-MAX sono unità di accesso ad un mezzo trasmissivo EHSA, che consentono ad un PC di essere connesso a detto bus.
Il pilotaggio di una scheda HS-MAX con PC avviene per via seriale, secondo un determinato protocollo, copyright della Società TRIALOG. La configurazione hardware del collegamento prevede l'utilizzo delle sole linee Tx, Rx e ground, una velocità di trasferimento pari a 19200 baud, parità pari, 1 bit di start e 1 bit di stop.
Lo schema di comunicazione prevede una struttura modulare a strati OSI-like. Sono previsti tre strati:



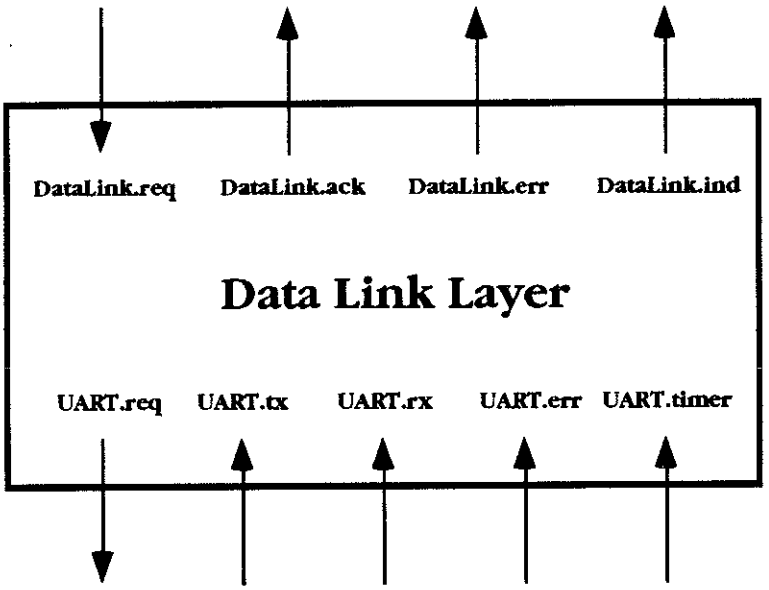
- Applicazione HS: Strato che realizza un nodo HS, che sfrutta i servizi forniti dalle schede HS-MAX. Tali servizi sono effettuati da alcune primitive che si occupano di scambiare PDU (Protocol Data Unit) con lo strato immediatamente inferiore (Data Link Layer).

- DataLink Layer: Strato intermedio tra l'*Applicazione HS* e l'*UART Management*, che si occupa di interfacciare questi ultimi.
- UART Management: È un piccolo starto che dipende strettamente dal tipo di interfaccia di comunicazione seriale residente sul PC. Esso tiene conto del tipo di UART (con o senza FIFO) e del modo con cui è interfacciata.

Di seguito sono descritti in maggiore dettaglio i due strati più bassi; lo strato relativo all'*applicazione HS* è ampiamente descritto nella nota HS-WinLib.

Data Link Layer.

La figura seguente schematizza i servizi forniti dal Data Link Layer ai due strati adiacenti.



- **DataLink.req**: Primitiva di richiesta per la trasmissione di un PDU. La lunghezza del PDU deve essere compresa nell'intervallo [1,125].
- **DataLink.ack**: Primitiva di acknowledgement, che indica il successo della trasmissione del pacchetto.
- **DataLink.err**: Primitiva di segnalazione di errore della trasmissione seriale.Il pacchetto non può essere trasmesso e viene saltato (passaggio all'eventuale successivo).
- **DataLink.ind**: Primitiva di indicazione di ricezione di un PDU. Come per la primitiva DataLink.req, il PDU deve avere una lunghezza compresa nell'intervallo[1,125].
- **UART.req**: Primitiva di richiesta di trasmissione di un byte sul collegamento seriale (verso la scheda HS-MAX).
- **UART.tx**: Primitiva di interrupt della trasmissione che segnala il successo della spedizione del byte.
- **UART.rx**: Primitiva di interrupt della ricezione che segnala il successo di ricezione di un byte.
- **UART.err**: Primitiva di interrupt di errore che segnala l'insuccesso della ricezione di un byte.

- **UART.timer**: Timer di interrupt di 1 msec.

Controllo del flusso

Il controllo del flusso dei dati è stato implementato con il meccanismo Xon/Xoff e segue il seguente principio di funzionamento:

- Il carattere Xoff è inviato non appena il buffer di ricezione raggiunge una data velocità di riempimento.
- Il carattere Xon è inviato non appena il buffer di ricezione raggiunge un'altra più bassa velocità di riempimento.
- Quando il carattere Xoff è ricevuto, la trasmissione è immediatamente fermata finché non viene ricevuto un nuovo carattere Xon. Il pacchetto correntemente trasmesso non andrà perduto.
- La trasmissione dei caratteri Xon, Xoff e dell'acknowledge è ancora permessa anche se è stato ricevuto un Xoff.

I caratteri Xon e Xoff hanno priorità sui pacchetti e sugli acknowledge. In tal modo possono essere ricevuti nel mezzo della trasmissione di un pacchetto o di un acknowledge.

Meccanismo di acknowledge.

Il meccanismo di acknowledge, realizzato per gestire gli errori di trasmissione, funziona secondo il seguente principio:

- Un pacchetto viene trasmesso solamente se quello precedente ha ricevuto l'acknowledge.
- Non appena un pacchetto è stato ricevuto, dovrebbe essere dato l'acknowledge. L'acknowledge contiene il numero del pacchetto, se questo è stato ricevuto senza errori ed ha un *buon formato*, o un numero di pacchetto non autorizzato, altrimenti.
- Tutti gli acknowledge ricevuti durante una trasmissione o non attesi, sono ignorati.
- Quando viene ricevuto un acknowledge con il numero di pacchetto giusto, il pacchetto è rimosso dal buffer e quello immediatamente successivo è subito trasmesso.
- Quando è ricevuto un acknowledge con il numero di pacchetto sbagliato (ovvero, diverso dal numero del pacchetto trasmesso), il pacchetto stesso viene trasmesso di nuovo.
- Se, dopo un certo intervallo di tempo dalla trasmissione di un pacchetto, non viene ricevuto alcun acknowledge, il pacchetto è trasmesso una nuova volta. Il numero di ritrasmissioni del pacchetto è limitato.
- Un pacchetto ritrasmesso senza successo tante volte fino al limite consentito viene rimosso dal buffer di trasmissione e l'applicazione riceve il messaggio di avvertimento di detta occorrenza.
- Un pacchetto ricevuto avente lo stesso numero di quello ricevuto immediatamente precedente con successo, viene soddisfatto dal meccanismo di acknowledge, ma è ignorato e l'applicazione non riceve alcuna segnalazione di ciò.

L'acknowledge ha priorità sui pacchetti trasmessi, di conseguenza può essere trasmesso o ricevuto nel mezzo della trasmissione di un pacchetto.

Formato del frame

Ogni pacchetto o acknowledge è terminato da un EVT (carattere di evento) per ragioni di compatibilit`à con il driver Windows. I caratteri che lo precedono servono ad individuare se il frame sia un pacchetto o un acknowledge.

- Un acknowledge consiste di 3 byte: il numero del pacchetto, il carattere di acknowledge ACK ed il carattere EVT.
- Un pacchetto inizia con il proprio numero e termina con la sua lunghezza, il carattere di 'fine pacchetto' EOP ed il carattere EVT.

I caratteri Xon, Xoff e di evento sono riservati e, come tali, non possono essere presenti in un pacchetto. Se, per qualsiasi ragione, debbano essere contenuti in un pacchetto, è necessario che siano preceduti dal carattere speciale Sp (che risulta anch'esso riservato), in modo da essere trasformati in caratteri non riservati.
Per la stessa ragione, il numero di pacchetto non può avere un valore corrispondente ad un carattere riservato.
La lunghezza del pacchetto spedito comprende tutti i caratteri speciali Sp e, se il suo valore corrisponde ad un carattere riservato, deve essere preceduto da Sp. I possibili caratteri Xon, Xoff e di acknowledge non sono considerati nel calcolo della lunghezza del pacchetto.

Esempio di pacchetto che contiene caratteri speciali

Si supponga di dovere trasmettere i seguenti dati

a	b	c	d	e	f

dove c = Xon ed e = Sp.
Il pacchetto sarà trasmesso come segue:

#	a	b	Sp	c*	d	Sp	e*	f		EOP	EVT

Nel frame precedente sono stati evidenziati con la bordatura sia i caratteri aggiunti che quelli alterati. In particolare, # indica il numero del pacchetto, mentre c* e e* sono i caratteri alterati corrispondenti a **c** e ad **e**. I caratteri alterati provengono dai caratteri originali differendo da questi per un offset costante predefinito: X* = X + Offset.
Nel caso che il carattere 12 (lunghezza del pacchetto) sia anch'esso speciale, allora il frame risulta modificato dal precedente:

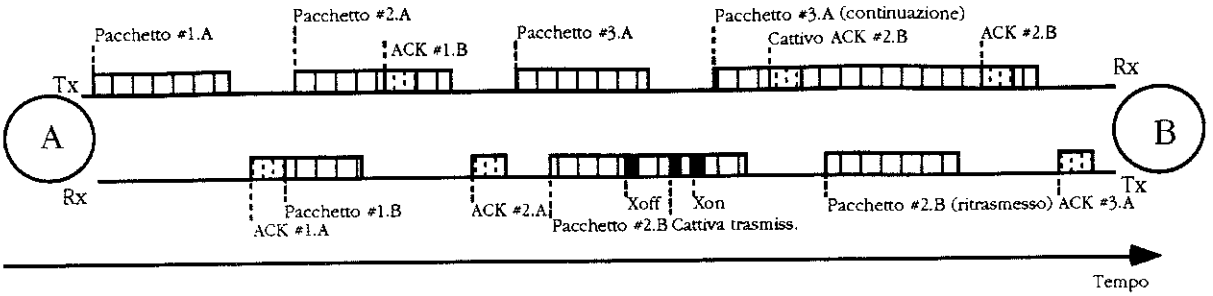
#	a	b	Sp	c*	d	Sp	e*	f	Sp		EOP	EVT

Nel caso che si sovrappongano, ad esempio, un acknowledge e un Xoff, il pacchetto risulta così frammentato:

#	a	b	Sp	c*	#	ACK	EVT	d	Sp	e*	Xoff	f	Sp		EOP	EVT

Esempio di comunicazione seriale che adotta il protocollo descritto.

La figura seguente mostra un diagramma temporale di alcune situazioni che possono presentarsi in una comunicazione tra due unità che sfruttano il protocollo visto



- L'unità A trasmette il Pacchetto #1.A.
- L'unità B risponde con l'acknowledge ACK #1.A ed invia immediatamente il Pacchetto #1.B.
- L'unità A risponde con l'acknowledge ACK #1.B nel mezzo della trasmissione del Pacchetto #2.A, che aveva intrapreso nel frattempo.
- L'unità B risponde alla A inviandole l'acknowledge ACK #2.A e quest'ultima trasmette il Pacchetto #3.A.
- Durante il trasferimento dati del Pacchetto #3.A, l'unità B, che in quel frangente aveva iniziato a trasmettere il Pacchetto #2.B, trova pieno il buffer di ricezione ed invia un Xoff all'unità A, che a sua volta interrompe la trasmissione del proprio pacchetto.
- Non appena il buffer di ricezione risulta svuotato, l'unità B invia un Xon, in modo che l'unità A continui la trasmissione del Pacchetto #3.A.
- Durante la trasmissione del Pacchetto #2.B, si verifica un errore nel trasferimento dei dati, pertanto l'unità A risponde a tale evento con un Cattivo ACK #2.B, invitando così l'unità B a ritrasmettere tale pacchetto.
- I due ultimi acknowledge (ACK #2.B e ACK #3.A) schematizzati in figura, di risposta ai reciproci pacchetti iniviati dalle due unità, sono di ovvia interpretazione.

Appendice A

Funzioni HS-WinLib

Di seguito sono riportate le descrizioni dei prototipi delle funzioni relative al pacchetto HS-WinLib. Per informazioni più dettagliate a riguardo, si consiglia di consultare lo User's Manual originale fornito dalla Società che detiene il copyright, della quale riportiamo l'indirizzo:

TRIALOG
rue du Château d'Eau
Paris, France

```
void AbandonToken (tApplicationId Id, tUInt8 Priority);
```

Primitiva usata daun API FC per rilasciare il token dopo averlo usato. Si dovrebbe attendere l'evento **cTokenPassingEvent**.

```
void RequestToken (tApplicationId Id, tUInt8 Priority);
```

Primitiva usata daun API FC quando vuole ottenere il token di un CoD. Si dovrebbe attendere l'evento **cTokenPassingEvent**.

```
void RetainToken (tApplicationId Id, tUInt8 Priority);
```

```
void SuspendToken (tApplicationId Id, tUInt8 Priority);
```

```
BOOL TP_ManageControlCommand(tApplicationId Id);
```

Primitiva usata da un CoD che utilizza il token passing per chiedere al modulo server del meccanismo in questione se un ad FC è consentito eseguire il comando o meno.

```
BOOL DefineEvent(tUInt8 Index, tUInt8 FAR *pData, tUInt8 Length);
```

Primitiva usata da unAPI CoD per consentire al modulo gestore degli eventi di esguire le notifiche degli eventi agli FC che sono interessati a quel particolare oggetto EHS.

```
BOOL EventManagementSignal(tUInt8 Index);
```

Primitiva usata da unAPI CoD per informare il modulo gestore degli eventi che un valore di un oggetto EHS è variato. Il nuovo valore sarà notificato a tutti gli FC interessati.

```
int GetCoD_Index(tApplicationId Id);
```

Primitiva usata da un API FC per rilevare l'indice nel database di enrolment (variabile **vEnrolledCoD**) che corrisponde all'ApplicationId (identificatore dell'API) di un CoD. Se l'Id non è nel database la funzione restituisce il valore -1.

```
int GetFC_Index(tApplicationId Id);
```

Primitiva usata da un API CoD per rilevare l'indice nel database di enrolment (variabile **vEnrolledFC**) che corrisponde all'ApplicationId di un FC. Se l'Id non è nel database la funzione restituisce il valore -1.

```
void InitTxBuffer(tEHS_Buffer *pBuffer);
```

Primitiva utilizzata per inizializzare un buffer EHS che sarà passato come parametro alla primitiva **SendCommand** o **SendResponse**. Essa riempie il buffer con valori di default. È obbligatorio usare questa primitiva se non tutti i campi del buffer sono riempiti dall'applicazione.

```
tStatus ScheduledDisconnection(void);
```

Primitiva utilizzata da un API per segnalare la propria rimozione (ess.: un FC segnala la proria rimozione ai CoD enrolled; Un CoD segnala la propria rimozione agli FC a cui è enrolled). Il comando inviato è un comando di gruppo.

```
void SendCommand(tEHS_Buffer Buffer);
```

Primitiva di richiesta con cui l'applicazione trasmette un comando. Nel caso che il comando non sia l'ultimo di una serie in un unico frame (meccanismo di blocking - parametro **consecutive** è TRUE), il processo applicativo deve mantenere il controllo e trasmettere gli altri comandi. Dopo avere trasmesso l'ultimo comando, la macchina a stati può aspettare l'evento **cLocalCnfEvent** per testare il successo o meno della trasmissione.

```
void SendDD_Read(void);
```

Primitiva usata da unAPI FC per provare ad effettuare l'enrolment dei nuovi CoD o dei CoD con Device Descriptor che in precedenza non era stato di interesse. Questa funzione è comunque eseguita ogni quarto d'ora dalle HS_WinLib.

`void` **`SendL7Cnf`**`(tUInt8` **`Result`**`);`

Primitiva di risposta a comandi non automaticamente confermati. Nel caso di risposta ad una serie di comandi in un unico frame (meccanismo di blocking - parametro **`consecutive`** è `TRUE`), l'applicazione deve mantenere il controllo e trasmettere le risposte successive.

`void` **`SendResponse`**`(tEHS_Buffer` **`Buffer`**`);`

Primitiva di risposta a comandi automaticamente confermati. Nel caso di risposta ad una serie di comandi in un unico frame (meccanismo di blocking - parametro **`consecutive`** è `TRUE`), l'applicazione deve mantenere il controllo e trasmettere le risposte successive.

`void` **`StartRegistration`**`(tUInt8` **`RegistrationCategory`**`, tUInt8` **`Address`**`);`

Primitiva usata dall'applicazione per avviare il processo di registrazione. Il secondo parametro è usato solamente per la registrazione di unità di categoria 3 (ciò, infatti, richiede che tali device scelgano il loro indirizzo).

`void` **`WaitEvent`**`(tUInt8` **`EventList`**`);`

Primitiva chiamata per far sì che l'**`ApplicationSM`** attenda uno o più eventi. La lista di eventi attesi deve essere costruita con i valori degli eventi individuali. Per esempio, un'applicazione che attenda un'indicazione o una conferma locale, dovrà passare alla funzione il parametro (**`cIndicationEvent`** | **`cLocalCnfEvent`**). In generale, una lista di eventi deve essere costruita eseguendo l'OR bit a bit degli eventi predefiniti necessari all'applicazione.

`void` **`ReadUnitConf`**`(LPSTR` **`FileName`**`);`

Primitiva usata per leggere un file di configurazione di un device. Il file deve essere scritto nello stesso formato dei file Windows. La configurazione di un device consiste nell'inizializzare i campi del device descriptor relativo.

`void` **`WriteUnitConf`**`(LPSTR` **`FileName`**`);`

Primitiva utilizzata per memorizzare su file la configurazione corrente di un device. Il formato del file deve essere lo stesso descritto per la funzione **`ReadUnitConf`**.

```
void SetDD(tDeviceDescriptor DD);
```

Primitiva utilizzata da un'applicazione per settare il proprio device descriptor. Nessun device descriptor è settato di default. Se non viene chiamata questa primitiva, il modulo client dell'enrolmentsarà disabilitato e l'applicazione non si comporterà come un CoD. Tale funzione può essere usata solamente quando il client database dell'enrolment è vuoto (**NumEnrolledFCs = 0**).

```
void SetTP_Type(tUInt8 Char);
```

Questa primitiva è utilizzata dall'applicazione per settare il tipo di server token passing. Il parametro è un carattere che può essere 'A', 'B' o 'C'. Qualsiasi altro valore indica che il server token passing è disabilitato (situazione di default).

```
void SetUniqueCode(tUniqueCode UC);
```

Questa primitiva è utilizzata dall'unità durante l'inizializzazione per specificare il proprio unique code. La chiamata a tale primitiva deve essere effettuata prima della fase di registrazione.

```
BOOL AddDD(tDeviceDescriptor DD);
```

Primitiva usata da un'applicazione per aggiungere un device descriptor alla lista gestita dal modulo server dell'enrolment. La lista è vuota allo startup. Se nessun device descriptor è aggiunto alla lista, il modulo server dell'enrolment sarà disabilitato e l'applicazione non si comporterà come un FC. Se questa primitiva viene chiamata dopo che la procedura di enrolment allo startup è finita, dovrebbe essere effettuata una chiamata alla primitiva **SendDD_Read** in modo che tutti i CoD con il device descriptor aggiunto saranno enrolled.

```
BOOL AddEvent(tEvent Object);
```

Primitiva utilizzata dall'applicazione per aggiungere un oggetto EHS alla lista gestita dal modulo gestore degli eventi. La lista è vuota allo startup. Se nessun oggetto è aggiunto alla lista, il modulo gestore degli eventi sarà disabilitato.

```
BOOL AddTestCommand(tTestCommand TestCommand);
```

Primitiva usata da un'applicazione per aggiungere un comando di test alla lista gestita dal modulo server dell'enrolment. Essa è usata per l'enrolment esteso. La lista è vuota allo startup. Se nessun comando di test è aggiunto alla lista, il modulo server dell'enrolment userà l'enrolment semplice.

`BOOL` **`CreateConsole`**`(BOOL` **`SystemInfo`**`);`

Primitiva utilizzata per creare una finestra per testo di I/O. Le stringhe sono scritte in tale finestra utilizzando la funzione **`WriteToConsole`**. Se il parametro passatole è `TRUE`, le informazioni sul sistema sono visualizzate sulla finestra aperta per aiutare il debugging dell'applicazione.

`void` **`CloseConsole`**`(void);`

Primitiva utilizzata per chiudere la fienstra di testo di I/O. Non è obbligatorio usarla, in quanto è comunque chiamata dalle HS-WinLib.

`BOOL` **`CreateRxBufferWatch`**`(void);`

Primitiva che consente all'utente di creare una finestra specifica in modo da monitorizzare l'attività del buffer di ricezione. Tutti i messaggi ricevuti dall'applicazione saranno visualizzati.

`BOOL` **`CloseRxBufferWatch`**`(void);`

Primitiva utilizzata per chiudere la fienstra RxBufferWatch. Non è obbligatorio usarla, in quanto è comunque chiamata dalle HS-WinLib.

`BOOL` **`CreateSM_Watch`**`(void);`

Primitiva che consente all'utente di creare una finestra specifica per monitorizzare gli eventi attesi e occorrenti della macchina a stati dell'applicazione.

`BOOL` **`CloseSM_Watch`**`(void);`

Primitiva utilizzata per chiudere la fienstra SM_Watch. Non è obbligatorio usarla, in quanto è comunque chiamata dalle HS-WinLib.

`BOOL` **`CreateTxBufferWatch`**`(void);`

Primitiva che consente all'utente di creare una finestra specifica in modo da monitorizzare l'attività del buffer di trasmissione. Tutti i messaggi ricevuti dall'applicazione saranno visualizzati.

```
BOOL CloseRxBufferWatch(void);
```

Primitiva utilizzata per chiudere la fienstra TxBufferWatch. Non è obbligatorio usarla, in quanto è comunque chiamata dalle HS-WinLib.

```
void WriteToConsole(LPSTR String);
```

Promitiva utilizzata per scrivere una stringa di testo su una finestra di I/O. Detta finestra dovrebbe essere aperta precedentemente con la funzione **CreateConsole**.

Appendice B

File header delle HS-WinLib

```
/*****************************************************************
* Copyright 1995
*    TRIALOG Informatique
*    9, Rue du Chateau d'eau    Tel : (33) 1 - 42 06 44 45
*    75010 PARIS
*    FRANCE
*
* Module         : ihs_win.h
* Version        : 1.0
* Project        : European Home Systems API for Windows
* Author         : Vincent Bouchy
* Creation date  : 19 12 94
* Document       :
* Description    : internal include file
*
*
*----------------------------------------------------------------
* Modifications :
* Author            Date        Description
*****************************************************************/

/* events for appli */
#define cInitEvent           0x00
#define cEndEvent            0xFF
#define cRegistrationEvent   0x01
#define cEnrolmentEvent      0x02
#define cIndicationEvent     0x04
#define cRemoteCnfEvent      0x08
#define cReceptionEvent      0x0C
#define cTokenPassingEvent   0x10
#define cLocalCnfEvent       0x20
#define cWindowsEvent        0x40
#define cNoEvent             0x00
#define cAllEvents           0xFF

/* Constants */
/*************/
#define cUnregistered    0x00
#define cConfirmeOld     0x01
#define cConfirmeNew     0x02

#define cNoConf          3
#define cConfOnFailure   2
#define cConfOnSuccess   1
#define cFullConf        0

#define cHighPriority    0x00
#define cNormalPriority  0x01
#define cLowPriority     0x03

#define cTP_TypeA    3
#define cTP_TypeB    1
#define cTP_TypeC    0
#define cNoTP        -1

/* TP_DB ClientState values */
#define cNotInterested  0
#define cGranted        1
```

```
#define cQueuing        2
#define cNotRequired    3
#define cRequired       4

/* from appli to TP module */
#define cInterestedWithToken              0
#define cInterestedWithTokenOrQueue       1
#define cFinishedWithToken                2
#define cRetain                           3
#define cSuspend                          4
/* from TP module to appli */
#define cFinishOrRetain                   5
#define cFinish                           6
#define cAuthorisation                    7
#define cAbandoned                        8
#define cNotAbandoned                     9
#define cRefused                          10
#define cQueued                           11
#define cTooManyControlledDevice          12
#define cNothing                          13

/* TP priorities */
#define cIdlePriority       0x60
#define cBackgroundPriority 0x62
#define cUserPriority       0x64
#define cUserPlusPriority   0x65
#define cTimerPriority      0x68
#define cUrgentPriority     0x6A
#define cSecurityPriority   0x6C
#define cSafetyPriority     0x6E
#define cInstallerPriority  0x6F

/* type definitions */
/*********************/
typedef BYTE  tApplicationId;

typedef struct {
BYTE  Id;
BYTE  Permanent;
BYTE  ObjectTS;
BYTE  ObjectOPC;
BYTE  Service;
BYTE  ConfType;
BYTE  InvokeId;
BYTE  Action;
BYTE  Consecutive;
BYTE  StopOnError;
BYTE  ServiceClass;
BYTE  SecurityRequested;
BYTE  RouteSecurity;
BYTE  Result;
BYTE  DataLength;
BYTE  Data[cMaxDataSize];
} tEHS_Buffer;

typedef struct {
BYTE  UniqueCode0;
BYTE  UniqueCode1;
BYTE  UniqueCode2;
BYTE  UniqueCode3;
BYTE  UniqueCodePresent;
} tUniqueCode;
```

```
typedef struct {
tApplicationId  Id;
BYTE          DD_TS;
BYTE          DD_OPC;
} tEnrolmentDB;

typedef struct {
BYTE            Event;
tApplicationId  Id;
} tApplicationDialog;

typedef struct {
BYTE  TS;
BYTE  OPC;
} tTS_OPC;

#define tDeviceDescriptor   tTS_OPC
#define tEvent              tTS_OPC

typedef struct {
BYTE  TS;
BYTE  OPC;
BYTE  Service;
} tTestCommand;

typedef struct {
BYTE FAR *lpData;
BYTE     Length;
} tEventData;

typedef struct {
tApplicationId        Id;
tUInt8                Priority;
tUInt8                State;
} tTP_DB;

typedef struct {
BOOL    fGo;
BOOL    fDialogBox;
BYTE    nComNumber;
int     nBaudRate;
} tAutoStart;

/* prototypes */
/**************/
int     PASCAL          WinMain (HANDLE, HANDLE, LPSTR, int);
static BOOL             InitApplication (HANDLE);
static BOOL             InitInstance (HANDLE, int);
long   FAR PASCAL   MainWndProc (HWND, UINT, WPARAM, LPARAM);
static void         InitializeComm(void);
static void             TreatBadMsg(void);
static void             EndAppli(void);
static BOOL             AnalysePacket(int, BYTE *);
static long             TreatWinAppliEvent(HWND, unsigned, WORD, LONG);
static void             InitHS_Variables(void);
BOOL   FAR PASCAL       About(HWND, unsigned, WORD, LONG);
BOOL   FAR PASCAL       Connect(HWND, unsigned, WORD, LONG);
BOOL   FAR PASCAL       ConfigDlgProc(HWND, unsigned, WORD, LONG);
BOOL                    CreateConsole(BOOL);
void                    CloseConsole(void);
BOOL                    CreateRxBufferWatch(void);
void                    CloseRxBufferWatch(void);
BOOL                    CreateTxBufferWatch(void);
void                    CloseTxBufferWatch(void);
```

```
BOOL                    CreateSM_Watch(void);
void                    CloseSM_Watch(void);
void                    WriteToConsole(LPSTR);
void                    InitTxBuffer(tEHS_Buffer *);
void                    StartRegistration(BYTE, BYTE);
void                    ScheduledDisconnection(void);
void                    WaitEvent(BYTE);
void                    SendCommand(tEHS_Buffer);
void                    SendResponse(tEHS_Buffer);
void                    SendL7Cnf(BYTE);
void                    RequestToken(BYTE, BYTE);
void                    RetainToken(BYTE, BYTE);
void                    SuspendToken(BYTE, BYTE);
void                    AbandonToken(BYTE, BYTE);
BOOL                    TP_Authorized(tApplicationId);
BOOL                    DefineEvent(BYTE, BYTE FAR *, BYTE);
void                    EventManagementSignal(BYTE);
BOOL                    AddDD(tTS_OPC);
BOOL                    RemoveDD(tTS_OPC);
void                    SendDD_Read(void);
void                    SetDD(tTS_OPC);
void                    SetUniqueCode(tUniqueCode);
BOOL                    AddTestCommand(tTestCommand);
BOOL                    AddEvent(tTS_OPC);
BOOL                    SetTP_Type(BYTE);
void                    WriteUnitConf(LPSTR);
void                    ReadUnitConf(LPSTR);
int                     GetFC_Index(tApplicationId);
int                     GetCoD_Index(tApplicationId);
extern void             ApplicationSM(void);

/* Windows constants */
/*********************/
#define csClassName  "EHS_WinAPI_WndClass"

/* menu */
#define IDM_ABOUT           100
#define ACTION_POS          1
#define IDM_CONNECT         110
#define IDM_DISCONNECT      111
#define WATCH_POS           2
#define CONSOLE_POS         0
#define IDM_SYSCONSOLE      200
#define IDM_CONSOLE         201
#define IDM_RXHIGH          121
#define IDM_TXHIGH          122
#define IDM_SMDESCR         123
#define IDM_CONFIG          103
#define IDM_EXIT            104

/* Connect Box */
#define ID_COM          666
#define ID_BAUDRATE     667
```

```
/*****************************************************************
* Copyright 1995 TRIALOG
*
* Developed, Supported and Maintained by:
*
*    TRIALOG Informatique
*    9, Rue du Chateau d'eau    Tel : (33) 1 - 42 06 44 45
*    75010 PARIS
*    FRANCE
*
* Module         : hsw_api.h
* Version        : 1.0
* Project        : HSWinLib
* Author         : Vincent Bouchy
* Creation date  : 21 12 94
* Document       :
* Description    : HS-WinLib API include file
*----------------------------------------------------------------
* Modifications :
* Author            Date        Description
*****************************************************************/

/* Constants */
/*************/
/* Events */
#define cInitEvent          0x00
#define cEndEvent           0xFF
#define cRegistrationEvent  0x01
#define cEnrolmentEvent     0x02
#define cIndicationEvent    0x04
#define cRemoteCnfEvent     0x08
#define cReceptionEvent     (cIndicationEvent + cRemoteCnfEvent)
#define cTokenPassingEvent  0x10
#define cLocalCnfEvent      0x20
#define cWindowsEvent       0x40
#define cNoEvent            0x00
#define cAllEvents          0xFF

/* Registration categories */
#define cRegCat1    0x00
#define cRegCat2    0x01
#define cRegCat3    0x02
#define cRegCatDAA  0x03

/* RegState variable values */
#define cUnregistered   0
#define cConfirmeOld    1
#define cConfirmeNew    2

/* Conf Type */
#define cNoConf         3
#define cConfOnFailure  2
#define cConfOnSuccess  1
#define cFullConf       0

/* Service Class */
#define cHighPriority   0
#define cNormalPriority 1
#define cLowPriority    3

/* ROM ATD entries */
#define cMDC_Entry      0
#define cAllFCsEntry    1
```

```
#define cAllCoDsEntry   2
#define cEveryBodyEntry 3

/* TP Events from module to appli */
#define cFinishOrRetain                 5
#define cFinish                         6
#define cAuthorisation                  7
#define cAbandoned                      8
#define cNotAbandoned                   9
#define cRefused                        10
#define cQueued                         11
#define cTooManyControlledDevice        12
#define cNothing                        13

/* TP priorities */
#define cIdlePriority       0x60
#define cBackgroundPriority 0x62
#define cUserPriority       0x64
#define cUserPlusPriority   0x65
#define cTimerPriority      0x68
#define cUrgentPriority     0x6A
#define cSecurityPriority   0x6C
#define cSafetyPriority     0x6E
#define cInstallerPriority  0x6F

/* TP_DB ClientState values */
#define cNotInterested  0
#define cGranted        1
#define cQueuing        2
#define cNotRequired    3
#define cRequired       4

/* EHS_Buffer Result values */
#define cSuccessfullyTransmitted  0x00
#define cServiceSuccessful        0x01
#define cServiceWouldSucceed      0x02
#define cL7_CantSend              0x04
#define cNoResponseEndUnit        0x20
#define cShortMT_PDU_Only         0x21
#define cN_PDU_TooLong            0x22
#define cUnableToSend             0x23 /* HS spec 1.1 extension -> for DAA
*/
#define cRouteOverloaded          0x30
#define cRouteDisabled            0x31
#define cRouteNotAvailableRouter  0x32
#define cRouteNotAvailableNetwork 0x33
#define cBadRoute                 0x35
#define cCommandBadlyFormatted    0x70
#define cDSUA_NotUnderstood       0x73
#define cObjectNotUnderstood      0x74
#define cServiceNotUnderstood     0x75
#define cInvalidArrayIndex        0x76
#define cDataOutOfRange           0x77
#define cInsecureDatagram         0x78
#define cEnrolmentRequired        0x7D
#define cTokenRequired            0x7E
#define cUnableExecuteNow         0x7F
#define cRequestedRouteNotSecure  0x81
#define cCouldNotBlock            0x82
#define cObjectNotUnderstoodOrig  0x84
```

```
/* Types */
/*********/
typedef BYTE    tUInt8;
typedef tUInt8  tApplicationId;

#define cMaxDataSize    256

typedef struct {
tUInt8  Id;
tUInt8  Permanent;
tUInt8  ObjectTS;
tUInt8  ObjectOPC;
tUInt8  Service;
tUInt8  ConfType;
tUInt8  InvokeId;
tUInt8  Action;
tUInt8  Consecutive;
tUInt8  StopOnError;
tUInt8  ServiceClass;
tUInt8  SecurityRequested;
tUInt8  RouteSecurity;
tUInt8  Result;
tUInt8  DataLength;
tUInt8  Data[cMaxDataSize];
} tEHS_Buffer;

typedef struct {
tUInt8  UniqueCode0;
tUInt8  UniqueCode1;
tUInt8  UniqueCode2;
tUInt8  UniqueCode3;
tUInt8  UniqueCodePresent;
} tUniqueCode;

typedef struct {
tApplicationId  Id;
tUInt8          DD_TS;
tUInt8          DD_OPC;
} tEnrolmentDB;

typedef struct {
tUInt8          Event;
tApplicationId  Id;
} tApplicationDialog;

typedef struct {
tUInt8  TS;
tUInt8  OPC;
} tTS_OPC;

#define tDeviceDescriptor   tTS_OPC
#define tEvent              tTS_OPC

typedef struct {
tUInt8  TS;
tUInt8  OPC;
tUInt8  Service;
} tTestCommand;

typedef struct {
tApplicationId      Id;
tUInt8              Priority;
tUInt8              State;
} tTP_DB;
```

```
typedef struct {
BOOL    fGo;
BOOL    fDialogBox;
BYTE    nComNumber;
int     nBaudRate;
} tAutoStart;

/* Variables */
/*************/
extern tUInt8                RegState;
extern int                   NumEnrolledCoDs;
extern int                   NumEnrolledFCs;
extern tEnrolmentDB          vEnrolledCoDs[];
extern tApplicationId        vEnrolledFCs[];
extern tApplicationDialog    ApplicationDialog;
extern tEHS_Buffer           RxBuffer;
extern BYTE                  Event;
extern HWND                  hMainWindow;
extern HINSTANCE             hInst;
extern UINT                  Win_message;
extern WORD                  Win_wParam;
extern LONG                  Win_lParam;
extern tUInt8                TxResult;
extern tTP_DB                vTokenPassingDB[];

/* Functions */
/*************/
extern BOOL CreateConsole(BOOL);
extern void CloseConsole(void);
extern BOOL CreateRxBufferWatch(void);
extern void CloseRxBufferWatch(void);
extern BOOL CreateTxBufferWatch(void);
extern void CloseTxBufferWatch(void);
extern BOOL CreateSM_Watch(void);
extern void CloseSM_Watch(void);
extern void WriteToConsole(LPSTR);

extern void InitTxBuffer(tEHS_Buffer *);
extern void StartRegistration(tUInt8, tUInt8);
extern void ScheduledDisconnection(void);
extern void WaitEvent(tUInt8);
extern void SendCommand(tEHS_Buffer);
extern void SendResponse(tEHS_Buffer);
extern void SendL7Cnf(tUInt8);
extern void RequestToken(tUInt8, tApplicationId);
extern void RetainToken(tUInt8, tApplicationId);
extern void SuspendToken(tUInt8, tApplicationId);
extern void AbandonToken(tUInt8, tApplicationId);
extern BOOL TP_Authorized(tApplicationId);
extern BOOL DefineEvent(tUInt8, tUInt8 FAR *, tUInt8);
extern void EventManagementSignal(tUInt8);
extern int  GetFC_Index(tApplicationId);
extern int  GetCoD_Index(tApplicationId);

extern BOOL AddDD(tDeviceDescriptor);
extern BOOL RemoveDD(tDeviceDescriptor);
extern void SendDD_Read(void);
extern void SetDD(tDeviceDescriptor);
extern void SetUniqueCode(tUniqueCode);
extern BOOL AddTestCommand(tTestCommand);
extern BOOL AddEvent(tEvent);
extern void SetTP_Type(tUInt8);
extern void WriteUnitConf(LPSTR);
```

```
extern void ReadUnitConf(LPSTR);

/* compatibility with HSLib */
/****************************/
#define cInit                       cInitEvent
#define cIndication                 cIndicationEvent
#define cRemoteCnf                  cRemoteCnfEvent
#define cReception                  cReceptionEvent
#define cLocalCnf                   cLocalCnfEvent

#define cNone                       cNoConf
#define cFail                       cConfOnFailure
#define cSucc                       cConfOnSuccess
#define cFull                       cFullConf

#define mEventTokenPassing    ApplicationDialog.EventTokenPassing
#define mControlledDevice     ApplicationDialog.ControlledDevice
#define mEventTokenPassingU   ApplicationDialog.EventTokenPassing
#define mControlledDeviceU    ApplicationDialog.ControlledDevice

#define mTokenState(i)              vTokenPassingDB[i].State
#define mTokenOwnerId(i)            vTokenPassingDB[i].Id

#define TP_ManageControlCommand(x)  TP_Authorized(x)
#define cAllowed                    TRUE
#define cNotAllowed                 FALSE

#define mRx_SourceId          RxBuffer.Id
#define mRx_Permanent         RxBuffer.Permanent
#define mRx_Object_TS         RxBuffer.ObjectTS
#define mRx_Object_OPC        RxBuffer.ObjectOPC
#define mRx_Service           RxBuffer.Service
#define mRx_ConfType          RxBuffer.ConfType
#define mRx_InvokeId          RxBuffer.InvokeId
#define mRx_Action            RxBuffer.Action
#define mRx_Consecutive       RxBuffer.Consecutive
#define mRx_StopOnError       RxBuffer.StopOnError
#define mRx_ServiceClass      RxBuffer.ServiceClass
#define mRx_SecurityReq       RxBuffer.SecurityRequested
#define mRx_RouteSecurity     RxBuffer.RouteSecurity
#define mRx_Result            RxBuffer.Result
#define mRx_DataLength        RxBuffer.DataLength
#define mRx_Data(i)           RxBuffer.Data[i]
#define mTx_SourceId          TxBuffer.Id
#define mTx_Object_TS         TxBuffer.ObjectTS
#define mTx_Object_OPC        TxBuffer.ObjectOPC
#define mTx_Service           TxBuffer.Service
#define mTx_ConfType(i)       (TxBuffer.ConfType = i)
#define mTx_InvokeId(i)       (TxBuffer.InvokeId = i)
#define mTx_Action(i)         (TxBuffer.Action = i)
#define mTx_Consecutive       TxBuffer.Consecutive
#define mTx_StopOnError(i)    (TxBuffer.StopOnError = i)
#define mTx_ServiceClass      TxBuffer.ServiceClass
#define mTx_SecurityReq       TxBuffer.SecurityRequested
#define mTx_RouteSecurity     TxBuffer.RouteSecurity
#define mTx_Result            TxBuffer.Result
#define mTx_DataLength        TxBuffer.DataLength
#define mTx_Data(i)           TxBuffer.Data[i]
```

```
/*****************************************************************
* Copyright 1994 by TRIALOG Informatique
*
*    TRIALOG Informatique
*    9, Rue du Chateau d'eau    Tel : (33) 1 - 42 06 44 45
*    75010 PARIS
*    FRANCE
*
* Module         : HS_CL.H
* Version        : 1.3
* Project        : Home Systems
* Author         : Olivier Marbach, Sylvain Sauvage
* Creation date  : 27/01/1993
* Document       : Home Systems Specification - Release 1.1
* Description    : Definition of command language objects values
*----------------------------------------------------------------
* Modifications :
* Author            Date        Description
*
*****************************************************************/


/************/
/* services */
/************/

/* Unconfirmed Services */
/************************/

/* Variables Access Services */
#define cWRT        0x00
#define cINF        0x20

/* Event Management Services */
#define cEVE        0x21
#define cEVD        0x22
#define cEVN        0x23

/* Domain Management Services */
#define cRDD        0x24
#define cINI        0x25
#define cTRM        0x26

/* Program Invocation Services */
#define cSTR        0x01
#define cCAN        0x02
#define cRUN        0x03
#define cEND        0x27
#define cFAIL       0x28
#define cDEC        0x10
#define cINC        0x11

/* Confirmed Services */
/**********************/

/* Variables Access Services */
#define cRD         0x40
#define cIDX        0x42

/* Domain Management Services */
#define cSEG        0x41
```

```
/************/
/* Objects */
/************/

/* Application Management - Init - Enrolment */
/*********************************************/
#define cAM_InitTS                              0x48

#define cDeviceDescriptorOPC                    0xE8

/* Application Management - General */
/************************************/
#define cAM_GeneralTS                           0x40

#define cNopOPC                                 0xCF

/* All Areas - General  */
/************************/
#define cAA_GeneralTS                           0x01

#define cPowerOPC                               0xC0
#define cBusControlEnableOPC                    0xC1

#define cBatteryStatusOPC                       0xD0
#define cPowerSourceOPC                         0xD1
#define cEndTimeOPC                             0xD2

/* All Areas - Timer */
/*********************/
#define cAA_TimerTS                             0x11

#define cRealTimeOPC                            0xC0
#define cRealDateOPC                            0xC1


/* HouseKeeping - General       */
/********************************/
#define cHK_General_TS                          0x06

#define cProgramActivationOPC                   0xD0
#define cStatusRequestOPC                       0xD1

/* HouseKeeping - Load Management  */
/***********************************/
#define cHK_LMM_TS                              0x3E

#define cPowerDemandOPC                         0xC0
#define cTotalPowerOPC                          0xD0 /* A_Sensor */
#define cCurrentTariffPeriodOPC                 0xE1


/* HouseKeeping - Lighting  */
/****************************/
#define cLightingTS                             0x1E

#define cLevelControlOPC                        0xC0
#define cLevelSensingOPC                        0xC1


/* HouseKeeping - Heating */
/**************************/
#define cHK_HeatingTS                           0x16

#define cRatedFlowTempOPC                       0xC0
```

```
#define cCurrentFlowTempOPC                 0xC1
#define cCurrentOutdoorTempOPC              0xC5
#define cRatedRoomTempOPC                   0xC8
#define cCurrentRoomTempOPC                 0xC9
#define cModeOfRoomCtrlOperationOPC         0xD7
#define cKeyCombinationOPC                  0xDA
#define cControlParameterOPC                0xDB


/**********************/
/* Device Descriptor */
/**********************/

/* Macros to split or join bytes of device descriptors */
#define mDD(a,b) (((tUInt16)a << 8) + (tUInt16)b)

#define mDD0(d)  ((tUInt8) (d >> 8))
#define mDD1(d)  ((tUInt8) (d & 0xFF))


/* Telecommunications */

#define cISDNGateway            0x400
#define cDChannel               0x401
#define cB1Channel              0x402
#define cB2Channel              0x403

#define cPSTNGateway            0x410
#define cVoiceTransceiver       0x412
#define cDTMFControl            0x413
#define cVoiceControl           0x414
#define cHayesCompatible        0x420
#define cNonHayesCompatible     0x421

#define cCTTerminal             0xC00
#define cKeyboardDisplay        0xC01
#define cAudioPort              0xC02
#define cDataPort               0xC03

#define cTelephone              0xC11
#define cAnsweringMachine       0xC12
#define cFaxMachine             0xC13
#define cAnalogueAdaptor        0xC14

#define cCallProgressToneGenerator  0xC21

/* Meter Reading */
#define cMeterWater             0x1C01
#define cMeterElectricity       0x1C02
#define cMeterGas               0x1C03

/* Audio/Video */

/* Security */

/* Safety */
#define cCurrentLeakSensor      0x2631
#define cCurrentSensor          0x2632
#define cCircuitBreaker         0x2633
#define cFireSmokeDetector      0x2641
#define cFireHeatDetector       0x2642
#define cSprinkler              0x2643
#define cGasLeakSensor          0x2651
#define cGasValve               0x2652
```

```
#define cWaterLeakSensor      0x2661
#define cWaterValve           0x2662
#define cEarthQuakeSensor     0x2671

/* White goods */
#define cWashingMachine                0x0E01
#define cDishWasher                    0x0E02
#define cMicroWave                     0x0E03
#define cOvenCavity                    0x0E04
#define cHob                           0x0E05
#define cHobHeater                     0x0E06
#define cHood                          0x0E07
#define cDryer                         0x0E08
#define cFridge                        0x0E09
#define cRegrigeratorCompartment       0x0E0A
#define cFreezeCell                    0x0E0B
#define cFoodTemperatureSensor         0x0E0C


/* Air conditioning */
#define cHumiditySensor                0x1601
#define cContaminationSensor           0x1602
#define cAirConditionner               0x1603
#define cCondensationSensor            0x1604
#define cOxygenLackSensor              0x1605
#define cCO2Sensor                     0x1606

/* Heating */
#define cRoomTemperatureSensor         0x1611
#define cExternalTemperatureSensor     0x1612
#define cExtractorFan                  0x1613
#define cPump                          0x1614
#define cMotorisedValve                0x1615
#define cBoilerController              0x1616
#define cHeatingCoordinator            0x1617
#define cMixingValveController         0x1618

/* Hot water */
#define cHotWaterTemperatureSensor     0x1621
#define cBathLevelSensor               0x1622
#define cHotWaterElectricHeater        0x1623
#define cBathTemperatureSensor         0x1624
#define cHotWaterMotorValve            0x1625
#define cBathWaterValve                0x1626
#define cHotWaterGasHeater             0x1627 /* !! Trialog added */

/* Noise */
#define cNoiseSensor    0x1631
#define cMailboxSensor  0x1632
#define cBell           0x0601

/* Lighting */
#define cLightOnOff     0x1E01
#define cLightDimable   0x1E02
#define cLightSensor    0x1E03

/* Electric lock */
#define cElectricKey          0x2E01
#define cWindowLockActuator   0x2E02
#define cDoorLockActuator     0x2E03

/* Motor system */
#define cMotor            0x3601
```

```
/* Load management */
#define cLMMcontroller    0x3E01

/* User interface */
#define cUBUI             0x1701
#define cPortableOneWay   0x0701
#define cPortableTwoWay   0x0702
#define cBasicUIinput     0x0F11
#define cBasicUIdisplay   0x0F12

#define cEnergySwitch             0x0711
#define cBellPush                 0x0712
#define cLightSwitchOnOff         0x0713
#define cLightSwitchDimmer        0x0714
#define cWindowLockSwitch         0x0715
#define cDoorLockSwitch           0x0716
#define cMotorSwitch              0x0717

#define cVoiceSynthesizer         0x0F01
#define cVoiceRecognitionDevice   0x0F02
#define cConnectedServiceRouter   0x0911

/********/
/* Data */
/********/

/* Choice Data Type */
#define cOFF 0x60
#define cON  0x61
#define cMIN 0x70
#define cMAX 0x71
#define cSTD 0x7F
```

Appendice C

Esempi di gestione software relativo al protocollo PC⇔HS-MAX.

Le costanti predefinite relative al protocollo descritto sono:

```
#deflne cRepeatMax   5
#deflne cAckTlmeOut  100

/* reserved character */

#deflne cXOn_Char        Ox11  /t Xon */
#deflne cXOff_Char       Ox13  /* Xoff */
#define cEvt_Char        0x16  /* event character */
#deflne cSp_Char         0x15  /* special chararcter */
#deflne cBad_Ack         0x14  /* bad packet number */
#define cCha_OffseMin    Ox11  /* lowest reserved character */
#define cChar_OffsetMax 0x17   /* highest reserved character + 1 */

/* packet ldentification characters */

#define cAck Char        0x06  /* end of ack datagram */
#define cEop_Char        ox21  /* end of packet datagram */
```

Di seguito è riportato un insieme di variabili globali utilizzate nei successivi spezzoni di codice C-like.

```
AckAwaited              // TRUE: acknowledgw atteso.
AckNumber               // Numero di pacchetto per l' acknowledge.
AckToSend               // Byte da spedire nell'acknowledge.
                        // (0 = nothing, 1 = #, 2 = ACK, 3 = EVI).
BytesSent               // Contatore dei byte inviati.
LengthSent              // TRUE: il byte 'lunghezza' è stato spedito nel
                        // pacchetto trasmesso.
NewPacket               // TRUE: nessun pacchetto è stato trasmesso.
NumberOfReceivedBytes   // Contatore dei byte ricevuti.
PacketCount             // Numero del pacchetto successivo da spedire.
ReceivedPacket          // Numero dell'ultimo pacchetto ricevuto.
ReceptionError          // TRUE: si è verificato un errore di ricezione e
                        // non è stato inviato il relativo acknowledge.
RepeatCount             // Contatore di rpetizioni del pacchetto trasmesso.
TimeCount               // Intervallo di tempo trascorso dall'ultimo
                        // pacchetto spedito.
Transmitting            // TRUE: è atteso l'interrupt della primitiva
                        // UART.tx.
XonToSend               // TRUE: Xon dovrebbe essere spedito.
XoffReceived            // TRUE: Xoff ricevuto.
XoffSent                // TRUE: Xon spedito.
XoffToSend              // TRUE: Xoff dovrebbe essere ricevuto.
```

- Pseudo-codice della primitiva **DataLink.req**:

```
{
      <Memorizza il PDU nel buffer di trasmissione>;
}
```

- Pseudo-codice della primitiva **`UART.tx`**:

```
{
      if (XoffToSend && !XoffSent)
      {
            UART.req(Xoff);
            XoffToSend = FALSE;
            XoffSent = TRUE;
      }
      else if (XcnToSend && XoffSent)
      {
            UART.req(Xon);
            XonToSend = FALSE;
            XoffSent = FALSE;
      }
      else if (AckToSend == 1)
      {
            AckToSend++;
            UART.req(AckNunber);
      }
      else if (AckToSend == 2)
      {
            AckToSend++;
            UART.req(cAck_Char);
      }
      else if (AckToSend == 3)
      {
            AckToSend = 0;
            UART.req(cEvt_Char);
      }
      else
      {
            if (< c'è almeno un byte da trasmettere>)
            {
                  if (!XoffReceived && !AckAwaited)
                  {
                        if NewPacket
                        {
                              BytesSent = 0;
                              NewPacket = FALSE;
                              UART.req(PacketCount);
                        }
                        else if (<tutti i byte sono spediti>)
                        {
                        if (!LengthSent)
                        {
                              if (<(BytesSent + 4) è in
                                    [cChar_OffsetMin,cChar_OffsetMax]>)
                              {
                                    UART.req(cSp_Char)
                                    BytesSent +- cChar_OffsetMax + 1;
                                    SpCharSent = TRUE;
                              }
                              else
                              {
                                    UART.req(BytesSent + 4);
                                    LengthSent = TRUE;
                              }
                        }
                        else
                        {
                              UART.reg(cEvt_Char);
                              TimeCount = 0;
```

```
                        AckAwaited = TRUE;
                        NewPacket = TRUE;
                        LengthSent = FALSE;
                    }
                    else
                    {
                        BytesSent++;
                        if (<next_byte è in
                            [cChar_OffsetMin,cCharOffsetMax]>)
                        {
                            <next_byte += cChar_OffsetMax>;
                            UART.req(cSp_Char);
                        }
                        else
                        {
                            UART.req(next_byte);
                            <punta al byte successivo dal buffer di
                                trasmissione>;
                        }
                    }
                }
                else
                    Transmitting = FALSE;
            }
            else
                Transmitting = FALSE;
}
```

- Pseudo-codice della primitiva **`UART.rx`**:

```
{
    if (ReceivedByte == cXoffChar)
        XoffRecelved = TRUE;
    else if (RecelvedByte == cXon_Char)
        XoffReceived = FALSE;
    else if (ReceivedByte == cEvt_Char)
    {
        if (<sono stati ricevuti meno di 2 byte>)
        {
            SendBadAck;
            Flush Reception Buffer;
            NumberOfReceivedBytes = 0;
        }
        else
        {
            if (ultimo_byte_ricevuto == cAck_Char)
            {
                NumberOfReceivedBytes -= 2;
                <elimina l'ultimo byte dal buffer di ricezione>;
                if AckAwaited
                {
                    if (ultimo_byte_ricevuto == PacketCount)
                    {
                        if (++PacketCount == cChar_OffsetMin)
                            PacketCount = cChar_OffsetMax + 1;
                        RepeatCount = 0;
                        <elimina il primo pacchetto dal buffer
                            di trasmissione>;
                        DataLink.ack;
                    }
```

```
                    else
                    TestAck;
                    AckAwaited = FALSE;
              }
              <elimina l'ultimo byte dal buffer di ricezione>;
        }
        else if (ultimo_byte_ricevuto == cFop_ Char)
        {
              if (<sono stati ricevuti meno di 3 byte>)
              {
                    SendBadAck;
                    Flush ReceptiGn Buffer:
              }
              else if (!ReceptionError)
              {
                    <elimina l'ultimo byte dal buffer di
                          ricezione>;
                    if (NunberOfReceivedBytes ==
                          ultimo_byte_ricevuto - 1)
                    {
                          AckNumber = <primo byte nel buffer di
                                      ricezione>;
                          AckToSend++;
                          if (AckNumber == ReceivedPacket)
                                Flush Reception Buffer;
                          else
                          {
                                ReceivedPacket = AckNumber;
                                if (Reception Buffer length >=
                                            clnBufferStop)
                                      XoffToSend = TRUE;
                                DataLink.ind;
                                Flush Reception Buffer;
                          }
                    }
                    else
                    {
                          SendBadAck;
                          Flush Reception Buffer;
                    }
              }
              else
              {
                    SendBadAck;
                    Flush Reception Buffer;
              }
              NumberOfReceivedBytes = 0;
        }
        else
        {
              call SendBadAck;
              Flush Reception Buffer;
              NumberOfReceivedBytes 0;
        }
}
else
{
        NuniberOfReceivedBytes++;
        if (ReceivedByte == cSp_Char)
              SpCharReceived = TRUE;
        else
        {
              if (SpCharReceived)
              {
```

```
                        ReceivedByte -= cChar OffsetMax;
                        SpCharReceived = FALSE;
                    }
                    <memorizza il byte ricevuto nel buffer di
                        ricezione>;
                    if (<il buffer di ricezione è pieno>)
                    {
                        Flush Reception Buffer;
                        ReceptionError = TRUE;
                    }
                }
            }
        }
}

Funzione TestAck()
{
        if (++RepeatCount == cRepeatMax)
        {
            RepeatCount = 0;
            DataLlnk.err;
            <elimina l'ultimo pacchetto dal buffer di trasmissione>;
            if (++PacketCount == cChar_OffsetMln)
                PacketCount = cChar_OffsetMax + 1;
        }
        else
            <punta al primo byte del buffer di trasmissione>;
}

Funzione SendBadAck()
{
        AckNunber = cBad_Ack;
        AckToSend++;
        ReceptlonError = FALSE;
}
```

- Pseudo-codice della primitiva **UART.err**:

```
{
        ReceptionError = TRUE;
}
```

- Pseudo-codice della primitiva **UART.timer**:

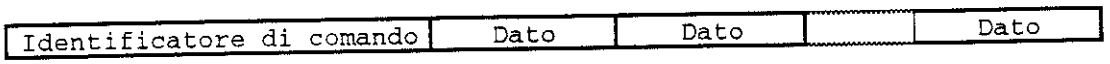
```
{
        if (AckAwaited)
        {
            if (++TimeCount == cAckTimeOut)
            {
                AckAwaited = FALSE;
                TestAck;
            }
        }
        if (!Transmitting)
        {
            Transmitting = TRUE;
            UART.tx;
        }
}
```

Appendice D

Descrizione del protocollo tra Applicazione HS e Data Link Layer

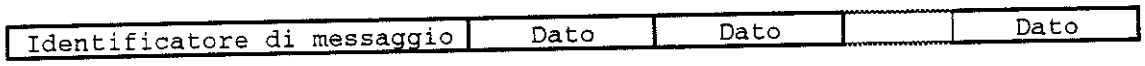
I PDU scambiati tra i due strati Applicazione HS e Data Link Layer di un sistema EHS che pilota le schede HS-MAX, possono essere descritti sinteticamente nel modo descritto di seguito:

- **Da PC a HS-MAX**: l'applicazione chiede alla scheda dei servizi EHS. Detti servizi sono richiesti tramite la primitiva **DataLink.req**. In tale direzione di trasferimento dati si distinguono i PDU **comandi** aventi il seguente formato



La definizione formale dei comandi è del tipo **cCmdxxx**, dove **xxx** è un codice mnemonico che indica il tipo di comando.

- **Da HS-MAX a PC**: la scheda informa il PC sullo stato dell'applicazione EHS. Le informazioni sono inviate tramite la primitiva **DataLink.ind**. In tale direzione di trasferimento dati si distinguono i PDU **messaggi** aventi il seguente formato

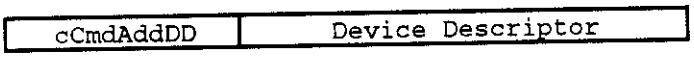


La definizione formale dei messaggi è del tipo **cMsgxxx**, dove **xxx** è un codice mnemonico che indica il tipo di comando.

Di seguito sono riportate le descrizioni dei singoli comandi e messaggi.

cCmdAddDD

Formato:



Lunghezza del comando: 3 byte.

Dato: DD

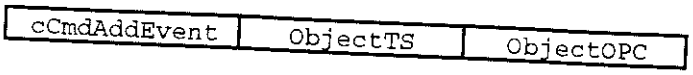
Device Descriptor da aggiungere (byte più significativo per primo).

Descrizione

Aggiunge un Device Descriptor alla lista usata dal modulo server dell'enrolment.

cCmdAddEvent

Formato:



Lunghezza del comando: 3 byte.

Dati:

`ObjectTS`

Table Selector dell'oggetto da aggiungere.

`ObjectOPC`

Opcode dell'oggetto da aggiungere.

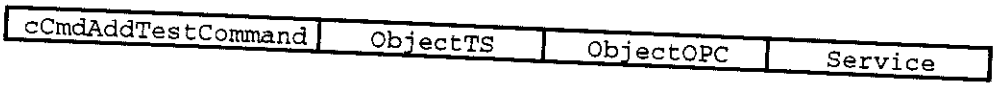
Descrizione

Aggiunge un oggetto alla lista usata dal modulo gestore degli eventi.

---

cCmdAddTestCommand

Formato:



Lunghezza del comando: 4 byte.

Dati:

`ObjectTS`

Table Selector dell'oggetto da aggiungere.

`ObjectOPC`

Opcode dell'oggetto da aggiungere.

`Service`

Servizio che deve essere provato sull'oggetto in questione.

Descrizione

Aggiunge un TestCommand alla lista usata dal modulo gestore degli eventi, per la procedura di enrolment esteso.

---

cCmdChangeDD

Formato:

cCmdChangeDD	Device Descriptor

Lunghezza del comando: 3 byte.

Dato: DD

Device Descriptor del device (byte più significativo per primo).

Descrizione

Cambia il Device Descriptor di un device e avvia il modulo client dell'enrolment. Non produce alcun effetto se il device è già enrolled da un FC.

---

cCmdChangeTP_Type

Formato:

cCmdCahnceTP_Type	TP_Type

Lunghezza del comando: 2 byte.

Dato: TP_Type

cTP_TypeA	0x03	Server Token Passing di tipo A
cTP_TypeB	0x01	Server Token Passing di tipo B
cTP_TypeC	0x00	Server Token Passing di tipo C
cNoTP	0xFF	Non c'è Server Token Passing

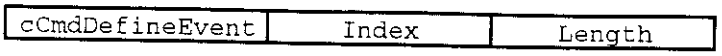
Descrizione

Cambia il tipo di modulo server token passing.

---

cCmdDefineEvent

Formato:



Lunghezza del comando: 3 byte.

Dati:

`Index`

Indice a base zero nella lista degli eventi usata dal modulo gestore degli eventi.

`Length`

Lunghezza del valore dell'oggetto.

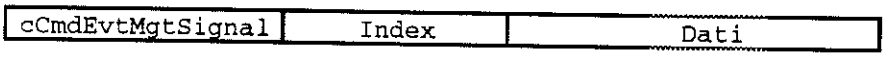
Descrizione

Segnala al modulo gestore degli eventi che il servizio EVE è adesso consentito sull'oggetto indicato dal parametro `Index` nella lista degli eventi riempita con il comando `cCmdAddEvent`. Il parametro `Length` sarà usato successivamente dal modulo gestore degli eventi per eseguire la notifica dell'evento (servizio EVN).

---

cCmdEvtMgtSignal

Formato:



Lunghezza del comando: almeno 3 byte.

Dati:

`Index`

Indice a base zero nella lista degli eventi usata dal modulo gestore degli eventi.

`Dati`

Nuovo valore dell'oggetto. La lunghezza di questo campo deve 'matchare' il valore del campo `Length` dato nel comando `cCmdDefineEvent`.

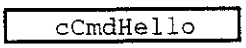
Descrizione

Aggiunge un oggetto alla lista usata dal modulo gestore degli eventi.

---

cCmdHello

Formato:



Lunghezza del comando: 1 byte.

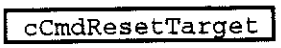
Descrizione

Chiede alla scheda HS-MAX di rispondere con un messaggio , per testare se la comunicazione seriale funziona correttamente.

---

cCmdResetTarget

Formato:



Lunghezza del comando: 1 byte.

Descrizione

Reinizializza la scheda HS-MAX (prima che la comunicazione seriale sia nuovamente ripristinata, deve trascorrere un corto intervallo di tempo dall'esecuzione del comando).

---

cCmdScheduleDiscnnection

Formato:

Lunghezza del comando: 1 byte.

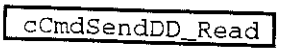
Descrizione

Avvia la sconnessione del device dal modulo server dell'enrolment.

---

cCmdSendDD_Read

Formato:



Lunghezza del comando: 1 byte.

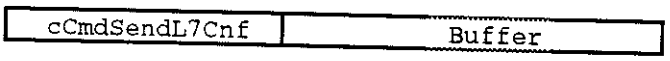
Descrizione

Questo comando fa sì che il modulo server dell'enrolment spedisca immediatamente un DD.RD sul bus EHS. È buona norma dopo l'esecuzione del comando `cCmdAddDD` effettuare immediatamente l'enrolment dei nuovi CoD.

---

cCmdSendL7Cnf

Formato:



Lunghezza del comando: 16 byte.

Dati:
`Buffer`

Buffer EHS lungo 15 byte, senza campo dati, la cui struttura è descritta nel comando `cCmdSendRequest`.

Descrizione

Trasmette sul bus EHS una risposta ad un servizio non automaticamente confermato.

---

cCmdSendRequest

Formato:

cCmdSendRequest	Buffer	Dati

Lunghezza del comando: almeno 16 byte.

Dati:

`Buffer`

Buffer EHS lungo 15 byte, senza campo dati, avente la seguente struttura:

```
typedef struct {
BYTE  Id;
BYTE  Permanent;
BYTE  ObjectTS;
BYTE  ObjectOPC;
BYTE  Service;
BYTE  ConfType;
BYTE  InvokeId;
BYTE  Action;
BYTE  Consecutive;
BYTE  StopOnError;
BYTE  ServiceClass;
BYTE  SecurityRequested;
BYTE  RouteSecurity;
BYTE  Result;
BYTE  DataLength;
} tBuffer;
```

`Dati`

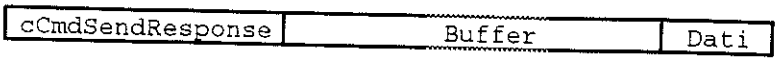
Dati per il buffer EHS la cui lunghezza è indicata nel campo `DataLength` del `Buffer`.

Descrizione

Spedisce una richiesta sul bus EHS.

cCmdSendResponse

Formato:



Lunghezza del comando: almeno 16 byte.

Dati:

`Buffer`

Buffer EHS lungo 15 byte, la cui struttura è descritta nel comando `cCmdSendResponse`.

`Dati`

Dati per il buffer EHS la cui lunghezza è indicata nel campo `DataLength` del `Buffer`.

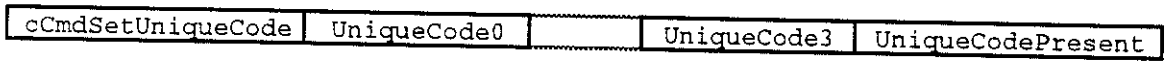
Descrizione

Trasmette sul bus EHS una risposta ad un servizio automaticamente confermato.

---

cCmdSetUniqueCode

Formato:



Lunghezza del comando: 6 byte.

Dati:

`UniqueCode0-3`

Byte che identificano lo Unique Code (non hanno alcun peso se risulta `UniqueCodePresent = FALSE`).

`UniqueCodePresent`

`TRUE`: i campi `UniqueCode0-3` definiscono lo Unique Code del device.

`FALSE`: non è previsto lo Unique Code per qul device.

Descrizione

Setta lo Unique Code del device (per default non è presente).

---

cCmdStartRegistration

Formato:

cCmdStartRegistration	Registration Category	Address

Lunghezza del comando: 2 o 3 byte.

Dati:

`Registration Category`

Questo byte serve a definire la categoria scelta per la procedura di registrazione.

cRegCat1	Registration Category 1
cRegCat2	Registration Category 2
cRegCat3	Registration Category 3
cRegCatDAA	Registration Category con DAA

`Address`

È un byte presente solamente per la registrazione di device appartenenti alla categoria 3 e contiene l'indirizzo al quale il device dovrebbe essere registrato.

Descrizione

Avvia la procedura di registrazione.

cCmdTokenPassing

Formato:

cCmdTokenPassing	TP_Command	Priority	CoD_Id

Lunghezza del comando: 4 byte.

Dati:

`TP_Command`

Tipo di comando Token Passing.

cTPCmdRequest	Richiede il Token
cTPCmdRetain	Ritiene il Token
cTPCmdSuspend	Sospende il Token
cTPCmdAbandon	Abbandona il Token

`Priority`

Priorità per la quale il Token è o deve essere mantenuto.

`CoD_Id`

Identificatore del CoD attualmente interessato al meccanismo di Token Passing..

Descrizione

Richiede, ritiene, sospende o abbandona il Token di un CoD. Il risultato del comando può poi essere letto nel messaggio `cMsgTokenPessingEvent` proveniente dalla scheda HS-MAX.

---

cMsgBadFormat

Formato:

cMsgBadFormat

Lunghezza del comando: 1 byte.

Descrizione

Segnala che l'unità non riconosce il formato del comando appena ricevuto.

---

cMsgEnrolmentClientDB

Formato:

cMsgEnrolmentClientDB	Index	Id

Lunghezza del comando: 3 byte.

Dati:

`Index`

Indice per il database del client per l'enrolment.

`Id`

Identificatore dell'applicazione dell'FC che effettua l'enrolment (-1 se un entry del database è vuoto).

Descrizione

Messaggio proveniente dal modulo client dell'enrolment. Segnala che un entry del corrispondente database è cambiato.

---

cMsgEnrolmentServerDB

Formato:

cMsgEnrolmentServerDB	Index	Device Descriptor	Id

Lunghezza del comando: 5 byte.

Dati:

`Index`

Indice per il database del server per l'enrolment.

`Device Descriptor`

Device Descriptor che ha subito l'enrolment.

`Id`

Identificatore dell'applicazione del CoD che ha subito l'enrolment (-1 se un entry del database è vuoto).

Descrizione

Messaggio proveniente dal modulo server dell'enrolment. Segnala che un entry del corrispondente database è cambiato.

---

cMsgHello

Formato:

cMsgHello

Lunghezza del comando: 1 byte.

Descrizione

Risposta al comando `cCmdHello`.

---

cMsgIndication

Formato:

cMsgIndication	Buffer	Dati

Lunghezza del comando: almeno byte.

Dati:

`Buffer`

Buffer EHS (15 byte) senza il campo dati, la cui struttura è definita nel comando `cCmdSendRequest`.

`Dati`

Dati per il buffer EHS, la cui lunghezza è indicata nel campo `Datalength` del `Buffer`.

Descrizione

Segnala un'indicazione dal bus EHS.

---

cMsgLocalCnf

Formato:

cMsgLocalCnf	Result

Lunghezza del comando: 2 byte.

Dati:

Result

cSuccessfullyTransmitted	La trasmissione ha avuto successo
cNoResponseEndUnit	L'unità remota non risponde
cUnableToSend	Il mezzo è occupato
cL7_CantSend	Il comando non ha il formato corretto

Descrizione

Segnala la fine della trasmissione di una richiesta o di una risposta.

Bibliografia

HS-WinLib User's Manual V1.00 - Home System Protocol Library - TRIALOG gennaio 1995.

Home System Specification Release 1.1 - EHSA 15 marzo 1992

Driving an HS-MAX - TRIALOG marzo 1995.

HS-MAX Serial Link: Data Link Layer Specification - TRIALOG marzo 1995.

HS-MAX Serial Link: Protocol Data Unit - TRIALOG marzo 1995.