Libreria C# per AD300A

Interfaccia via TCP con centralina antincendio

Manuale di riferimento DLL v1.0

manuale v1.0 - Enrico Colombini

Indice

1 Introduzione

La libreria ha il compito di interfacciare un programma con la centralina antincendio via seriale simulata su TCP su collegamento Ethernet, inserendosi nelle rete di centraline come **ripetitore**. Non invia comandi alla centralina, ma risponde ai comandi ricevuti e ricorda gli eventi segnalati dalla centralina stessa.

La libreria conserva in memoria una lista degli **eventi** più recenti ricevuti dalla centralina, fino al numero massimo specificato in fase di inizializzazione.

A scopo diagnostico è anche possibile **registrare** i pacchetti a basso livello trasmessi e ricevuti sulla linea di comunicazione (vedi "Packet logging").

Per "Manuale del protocollo AD300A" si intende il documento "Protocolo de comunicaciones RS-232/485 para las centrales AD300A de Advantronic Systems".

2 Componenti

La "solution" di Visual Studio contiene tre project:

- **FireAlarm**: la libreria DLL.
- **FireAlarmTest**: un eseguibile usato per i test durante lo sviluppo, utile come esempio di impiego della libreria.
- **CalcCrc**: una semplice utility per il calcolo del CRC dei pacchetti secondo le regole del produttore della centralina (non documentata qui).

La libreria `FireAlarm` è composta da diverse classi nel **namespace** `AD300A`.
Le classi pubblicamente accessibili dentro il namespace sono:

- **`FireAlarm`**: le API pubbliche.
- **`Event`**: la descrizione di un singolo evento.
- **`Packet`**: la descrizione di un singolo pacchetto dati.

Per informazioni sulle altre classi, vedi il capitolo "Note tecniche sulla struttura della libreria".

3 Uso della libreria

Questo capitolo è inteso come introduzione alle funzioni più comuni; per i dettagli e le rimanenti funzioni consultare il capitolo "Guida di riferimento delle API".

La libreria lavora esclusivamente in RAM senza eseguire scritture sul filesystem, pertanto può essere utilizzata anche su dispositivi embedded.

3.1 Avviamento

Per avviare il **thread di comunicazione** che si occupa autonomamente del collegamento con la centralina si crea un'istanza della classe `FireAlarm`:

```
fireAlarm = new AD300A.FireAlarm(fireAlarmCfg);
```

L'argomento fornito al costruttore di FireAlarm è un **dizionario** di coppie nome-valore, entrambi in formato di stringa (per comodità di lettura da file o altre fonti). Ad esempio:

```
Dictionary<string, string> fireAlarmCfg =
    new Dictionary<string, string>()
    {
        { "tcpAddress", "192.168.200.118" },
        { "tcpPort", "10001" },
        { "myFireNetAddress", "3" },
        { "aliveTimeoutSeconds", "60" },
        { "eventListSize", "1000" },
        { "packetLogSize", "1000" },
    };
```

Gli **argomenti** richiesti e i rispettivi valori sono descritti nella guida di riferimento. In caso di argomento mancante o non valido, il costruttore lancia una `ApplicationException`.

Durante il **normale funzionamento** della libreria non vengono inviate altre exception al programma ospite, a meno di errori nel codice.

3.2 Configurazione della centralina antincendio

È importante assicurarsi che l'**indirizzo** della libreria nella rete delle centraline, indirizzo indicato dal parametro `myFireNetAddress` nel dizionario passato al costruttore di `FireAlarm`, sia **univoco**.

Una volta avviata la libreria, la centralina va impostata per eseguire una **ricerca automatica** delle centrali collegate. Se tutto funziona correttamente, la libreria verrà identificata come un **ripetitore** all'indirizzo impostato.

Importante: solo se il riconoscimento è avvenuto correttamente, la centralina sarà in grado di rilevare e segnalare l'eventuale assenza della libreria. È quindi necessario che la libreria sia **avviata** nel caso venissero eseguite successive ricerche automatiche sulla centralina, affinché venga nuovamente riconosciuta.

Qualora si volesse **cambiare** l'indirizzo della libreria nella rete delle centraline, sarà necessario eseguire due volte la ricerca automatica: una volta col cavo di rete staccato per far dimenticare l'indirizzo vecchio, la seconda col cavo connesso e la libreria riavviata con i nuovi parametri, per fare apprendere alla centralina quello nuovo.

3.3 Controllo dello stato della connessione

La proprietà **`LineIsAlive`** indica lo stato presunto della connessione. Più esattamente:

- Se `LineIsAlive` è `true`, la connessione TCP è **sicuramente attiva**.
- Se `LineIsAlive` è `false`, è possibile che il cavo sia disconnesso o che non siano stati ricevuti pacchetti dalla centralina per il **tempo massimo** impostato alla creazione (`aliveTimeoutSeconds`). In questo caso la libreria tenta di riconnettersi a intervalli di 3 secondi.

Nota: `LineIsAlive` indica la ricezione di pacchetti AD300A validi, ma **non garantisce** che la libreria sia riconosciuta dalla centralina (i pacchetti potrebbero essere destinati a un'altra centralina).

3.4 Ricezione degli eventi

La libreria riceve pacchetti dati dalla centralina, li esamina e li converte, se appropriato, in **eventi** che conserva una **lista** (buffer a rotazione), fino a un massimo impostato in fase di configurazione (`eventListSize`), superato il quale gli eventi più vecchi vengono sovrascritti.

Per ricevere gli ultimi eventi comunicati dalla centralina, o tutti gli eventi nel caso della prima chiamata, si usa **`GetUnreadEvents()`**:

```
AD300A.Event[] newEvents = fireAlarm.GetUnreadEvents();
```

Il valore ritornato è una **lista di eventi**, nell'ordine in cui sono stati inviati dalla centralina. La successiva chiamata ritornerà soltanto eventuali **nuovi eventi** pervenuti nel frattempo.

La `GetUnreadEvents()` va chiamata **periodicamente** per evitare di perdere eventi, il che può avvenire se il numero di eventi ricevuti dall'ultima chiamata supera la capacità impostata con `eventListSize`.
Nel caso ciò si verifichi, la proprietà `FireAlarm.DidLoseEvents` diventa `true`; ritorna a `false` solo quando viene letta.

La **guida di riferimento** vedi capitolo relativo) dettaglia come accedere alle informazioni contenute in un evento e come richiedere invece l'intera lista di eventi, oppure solo gli eventi dall'ultimo Reset.

Per motivi di efficienza gli eventi ritornati sono **reference** agli eventi contenuti nella lista della libreria; pertanto il contenuto degli eventi **non va modificato**.

Una **descrizione leggibile** dell'evento può essere ottenuta con `Describe()`.

Note:
- La lista degli eventi non viene azzerata in caso di riconnessione.
- Non è garantito che la lista degli eventi contenga un Reset.
- È possibile che uno stesso evento sia presente più volte nella lista.

3.5 Chiusura della libreria

Il thread di comunicazione con la centralina funziona in background e quindi **si chiude** da sé uscendo dal programma ospite.
Nel caso il programma si chiudesse in modo anomalo, la libreria ne **rileva** lo stato e chiude comunque il thread e la comunicazione TCP.

Qualora invece la libreria venisse **chiusa e ricreata** senza uscire dal programma, ad esempio in seguito a una exception (in caso di errore non previsto della libreria) è bene utilizzare `Dispose` per garantire la chiusura della porta di comunicazione:

```
Dispose(fireAlarm);
```

L'esecuzione di `Dispose`, o in genere la chiusura del thread, può richiedere fino a 4 secondi (worst case in caso di pacchetti danneggiati, assai poco probabile; solitamente basta una frazione di secondo).

3.6 Packet logging

A scopo diagnostico è possibile vedere i pacchetti di comunicazione **a basso livello**, impostando il packet logging nella modalità desiderata con `SetLogMode()`, come descritto nella guida di riferimento.

Il packet logging ha un **costo** in termini di memoria ed efficienza, specie nel caso si inseriscano nel log tutti i pacchetti indistintamente, dato che occorre creare copia di ogni pacchetto.

Per vedere il **contenuto del log** si usa `GetLoggedPackets()` che ritorna la lista dei pacchetti aggiunti al log dall'ultima chiamata:

```
AD300A.Packet[] newPackets = fireAlarm.GetLoggedPackets ();
```

Il **numero massimo** di pacchetti contenuti nel log è impostato in fase di creazione (`packetLogSize`).
Nel caso in cui `GetLoggedPackets()` non venga chiamata per troppo tempo e dei pacchetti vengano **sovrascritti** nel log, la proprietà `FireAlarm.DidLogLosePackets` diventa `true`; ritorna a `false` solo quando viene letta.

Il contenuto di un pacchetto può essere mostrato come **byte dump** con `ToString()` e come **descrizione leggibile** con `Describe()`.

4 Il programma di test

Il programma `FireAlarmTest`, utilizzato durante lo sviluppo, è fornito a titolo di **semplice esempio** per mostrare come utilizzare le funzioni più comuni della libreria. Non è un'applicazione completa e non è a livello di produzione (ad esempio, la configurazione è scritta nel programma stesso e la gestione delle zone è semplificata).

Può essere utilizzato per **test** e per **logging** di eventi e pacchetti.
Il programma presenta la lista degli eventi ricevuti e, opzionalmente, il log dei pacchetti, con la possibilità di salvare le liste su file per un successivo esame.
Di seguito sono descritte alcune particolarità, senza pretesa di completezza.

4.1 Operazioni periodiche

Ogni secondo il programma aggiorna lo stato (presunto) della **connessione** con la centralina, le informazioni sul **numero di eventi** presenti nella lista della libreria (senza leggerli) e, se abilitato, il **log** dei pacchetti.

Se **Auto-update** è abilitato, aggiorna e mostra periodicamente gli eventi letti dalla libreria; aggiorna inoltre lo stato delle zone (quest'ultima funzione è minimale e solo esemplificativa; tra l'altro il numero di zone è stabilito con la costante `numZones` e non tiene conto di eventuali disconnessioni).

4.2 La lista degli eventi

È possibile selezionare quali eventi vedere:

- **Tutti gli eventi** nella lista della libreria.
- Solo gli eventi **dall'ultimo Reset**, se presente in lista (altrimenti tutti).

I pulsanti svolgono le seguenti funzioni:

- **Update**: legge e mostra eventuali nuovi eventi ricevuti dall'ultima Update o Auto-update (Auto-update preme questo pulsante con l'intervallo specificato).
- **Clear**: cancella la lista di eventi mostrata; non altera quella della libreria.
- **Re-read**: legge nuovamente dalla libreria la lista del tipo selezionato; cambiare selezione (tutti o dall'ultimo Reset) esegue anche un Re-read.
- **Save**: consente di registrare la lista di eventi visibile, che non necessariamente coincide con quella della libreria, in un file testo.

In ogni caso la lista di eventi contenuta nella libreria **non può essere alterata** dal programma; i pulsanti agiscono solo sulla lista mostrata all'utente.

Qualora uno o più eventi fossero stati **persi** per non avere eseguito Update o Auto-update con sufficiente frequenza o per avere configurato la libreria con un valore di `eventListSize` troppo basso, viene mostrato un messaggio nel punto corrispondente della lista di eventi.

Le descrizioni degli eventi sono mostrate assumendo una codifica **ASCII** a 7 bit (modificabile in `fireAlarmEncoding`).

4.3 Il log dei pacchetti

È possibile selezionare la **modalità** desiderata di logging, tra quelle offerte da `SetLogMode()` e descritte nella guida di riferimento.

Describe logged packets consente di aggiungere una descrizione leggibile al semplice dump esadecimale.
Gli eventuali **campi descrittivi** contenuti nel pacchetto sono mostrati tali e quali e non vengono riordinati logicamente come avviene per gli eventi, per cui è possibile che una stringa lunga sia mostrata suddivisa in due parti.

I pulsanti svolgono le seguenti funzioni:

- **Clear log**: cancella la lista di pacchetti mostrata; a differenza di quanto avviene con gli eventi, non sarà più possibile leggere i pacchetti precedenti.
- **Save**: consente di registrare la lista di pacchetti visibile in un file testo.

Qualora uno o più pacchetti fossero stati **persi** per avere configurato la libreria con un valore di `packetLogSize` troppo basso, viene mostrato un messaggio nel punto corrispondente della lista di pacchetti.

Ogni variazione della modalità di packet logging ha effetto immediato.
Quando il packet logging è disabilitato, la libreria **non tiene copia** dei pacchetti e non sarà quindi possibile esaminarli in seguito, a differenza di quanto accade con gli eventi.

Nota sul doppio timestamp: per comodità di debugging, prima di ciascun pacchetto il programma di test mostra il timestamp **locale** del programma di test, ossia `DateTime.Now()` al momento del suo inserimento nella lista mostrata.
Se è richiesta la descrizione del contenuto dei pacchetti, l'eventuale timestamp originale **della centralina** è mostrato sulla seconda riga della descrizione, come "date:" e "time:".

L'eventuale descrizione dei pacchetti è mostrata assumendo una codifica **ASCII** a 7 bit (modificabile in `fireAlarmEncoding`).

4.4 Simulazioni

La funzione `SimulateEvents()`, fornita sotto forma di commento, mostra come creare **finti eventi** per testare la logica ad alto livello di un programma applicativo, senza bisogno del collegamento fisico con la centralina.

5 Note tecniche sulla struttura della libreria

La libreria comprende le seguenti classi, tutte nel namespace AD300A.

5.1 Classi pubblicamente accessibili

- **`FireAlarm`**: le API pubbliche.
- **`Event`**: un evento ricevuto dalla centralina.
- **`Packet`**: un pacchetto di comunicazione.

5.2 Classi per uso interno

- **`Monitor`**: il thread di comunicazione e il relativo controller. Tutte le operazioni interne della libreria avvengono in questo thread.
- **`Packet_comm:`** parte privata di `Packet per la` trasmissione e ricezione di un pacchetto di comunicazione.
- **`Protocol`**: logica di interpretazione dei pacchetti, creazione degli eventi ed eventuali risposte alla centralina.
- **`EventList`**: buffer rotante thread-safe degli eventi ricevuti.
- **`PacketLog`**: buffer rotante thread-safe dei pacchetti ricevuti, usato solo se il packet logging è attivo.
- **`Utils`**: funzioni accessorie.

6 Guida di riferimento delle API

Di seguito sono **dettagliati** proprietà, metodi e campi pubblicamente accessibili delle classi esposte dalla API (`FireAlarm`, `Event` e `Packet`).

I nomi sono nel **namespace** `AD300A` e nella **classe** indicata in testa a ciascuna tabella. Ad esempio, con `Version` nella tabella relativa alla classe `FireAlarm` si intende:

```
AD300A.FireAlarm.Version
```

Legenda:

- I nomi con iniziale maiuscola, parentesi ed eventuali argomenti sono **funzioni** (metodi). Esempio: `GetAllEvents()`
- I nomi con iniziale maiuscola senza parentesi sono **proprietà**, in genere a sola lettura. Esempio: `Version`
- I nomi con iniziale minuscola sono **campi**. Esempio: `loop`

Per i **tipi** dei parametri e dell'eventuale valore di ritorno (quando non indicati), fare riferimento ai sorgenti.

6.1 classe FireAlarm

`FireAlarm(cfg)`	**Crea un'istanza della libreria**, avvia il thread di comunicazione che cerca di connettersi o riconnettersi (all'infinito).
	
	`cfg` è un dizionario di **coppie nome-valore** che deve contenere i seguenti nomi (i valori sono esemplificativi):
	
	`{ "tcpAddress", "192.168.200.118" },`
	`{ "tcpPort", "10001" },`
	`{ "myFireNetAddress", "3" },`
	`{ "aliveTimeoutSeconds", "60" },`
	`{ "eventListSize", "1000" },`
	`{ "packetLogSize", "1000" },`
	
	I **range validi**, definiti nel costruttore, sono:
	
	`tcpPort: "1".."65535"`
	`myFireNetAddress: "1".."32"`
	`aliveTimeoutSeconds: "10".."600"`
	`eventListSize: "1".."100000"`
	`packetLogSize: "1".."100000"`
	
	In caso di valori mancanti o non validi, lancia **`ApplicationException`** con una descrizione del problema.

`Dispose()`	**Elimina l'istanza** della libreria, ferma il relativo thread, chiude la connessione TCP. La **chiusura** del thread può richiedere fino a 4 secondi (worst case, assai poco probabile).
`Version`	**Versione** della libreria, come stringa Il suo valore è definito in `AssemblyInfo.cs`

`LineIsAlive`	Stato della linea: `true`: connessione TCP **sicuramente attiva**. `false`: cavo disconnesso o nessun pacchetto ricevuto dalla centralina per il **tempo massimo** impostato alla creazione (`aliveTimeoutSeconds`). In questo caso la libreria tenta di riconnettersi a intervalli di 3 secondi. Nota: `LineIsAlive` indica il transito sulla linea di pacchetti AD300A validi, non necessariamente diretti alla libreria, quindi **non garantisce** che la libreria sia vista dalla centralina. Tuttavia, se correttamente configurata con l'autoricerca in fase di installazione, **la centralina** segnalerà l'eventuale impossibilità di comunicare con la libreria dopo circa due minuti (tempo rilevato sperimentalmente).
`TotalEventCount`	**Numero totale di eventi** contenuti nella lista della libreria, inclusi quelli già letti. Il valore parte da zero e arriva al massimo a `eventListSize` impostato alla creazione, dopo di che **resta costante** in quanto i nuovi eventi sostituiscono quelli più vecchi.
`UnreadEventCount`	Numero di eventi **non ancora letti** con `GetAllEvents()`, `GetUnreadEvents()` o `GetEventsFromReset()`.
`ResetEventCount`	Numero di eventi **dall'ultimo evento di Reset**, compreso. Se **non ci sono** eventi di Reset in lista, indica il numero totale di eventi contenuti nella lista.

`DidLoseEvents`	Indica se sono stati **persi degli eventi**, ossia se uno o più eventi sono stati sovrascritti da nuovi eventi prima di essere stati letti con `GetAllEvents()`, `GetUnreadEvents()` o `GetEventsFromReset()`. **Leggere** questa proprietà ne riporta il valore a `false`.
`DidLogLosePackets`	Indica se sono stati **persi dei pacchetti dal log**, ossia se uno o più pacchetti sono stati sovrascritti da nuovi pacchetti prima di essere stati letti con `GetLoggedPackets()`. Ha senso solo se il **packet logging** è attivo. **Leggere** questa proprietà ne riporta il valore a `false`.
`GetAllEvents()`	Ritorna una nuova lista di (reference a) **tutti gli eventi** presenti nella lista della libreria. I campi degli eventi **non vanno modificati**.
`GetUnreadEvents()`	Ritorna una nuova lista di (reference agli) **eventi non ancora letti** con `GetAllEvents()`, `GetUnreadEvents()` o `GetEventsFromReset()`. I campi degli eventi **non vanno modificati**.
`GetEventsFromReset()`	Ritorna una nuova lista di (reference agli) eventi **dall'ultimo evento di Reset**, compreso. Se **non ci sono** eventi di Reset in lista, ritorna tutti gli eventi contenuti nella lista. I campi degli eventi **non vanno modificati**.

`SetLogMode(logMode)`	Imposta la **modalità di packet logging**, che può essere cambiata in qualsiasi momento. I valori sono definiti in `enum LogModes:` `LogModes.`**`None`**: logging **disabilitato**, cancella anche il log corrente. `LogModes.`**`RxButStatus`**: log dei **pacchetti significativi** ricevuti, ossia esclusi quelli periodici di status check. `LogModes.`**`RxTx`**: log di tutti i pacchetti **ricevuti e trasmessi**. `LogModes.`**`AllNet`**: log di **tutti i pacchetti**, anche per altri destinatari nella rete delle centraline. **Nota**: il logging ha funzione di debug ed è relativamente poco efficiente; inoltre il passaggio a `LogModes.None` tiene fermo il thread di comunicazione per il tempo necessario a cancellare i pacchetti dalla lista.
`GetLoggedPackets()`	Ritorna una nuova lista contenente (reference ai) pacchetti **non ancora letti** in precedenza con la medesima chiamata.

6.2 classe Event

(note)	Un evento viene **creato** e aggiunto alla lista quando la libreria riceve un messaggio dei tipi sotto elencati nella descrizione del campo `etype`. I **numeri di pagina** indicati si riferiscono al manuale del protocollo AD300A. I **campi** dell'evento, sotto descritti, sono liberamente accessibili, ma non si devono **modificare** quelli degli eventi ricevuti dalla libreria. In caso di dubbio nell'**interpretazione dei valori** forniti dalla centralina, fare riferimento alle pagg. 7,8 (del manuale del protocollo).
`Event(eType, packet)`	**Crea un evento** del tipo indicato (vedi `etype` sotto) leggendo gli altri campi dal pacchetto `packet`. Nell'uso normale **non è necessario** creare eventi; questo costruttore è accessibile per debugging e simulazioni.

etype	**Tipo dell'evento.** I valori possibili, elencati in `enum ETypes,` sono: **Unknown** Valore di default, non usato negli eventi creati dalla libreria. Eventi **globali** (centrale = 0, gli altri campi non hanno significato), rif. pag.13: **Reset** **ActivateSirens** **SilenceSirens** **Accepted** (pulsante "tacita") **Evacuation** Eventi **individuali** (con centrale, loop, zona, indirizzo, timestamp e altri dati), rif. pag. 3 da n.16 a n.20: **Fault** **Disconnection** **PreAlarm** **Alarm** **ButtonAlarm** Gli altri tipi di messaggi ricevuti dalla centralina **non generano eventi**.
central	Numero della **centrale**.

loop	Numero di **loop** dell'elemento.
	Casi particolari (rif. pag. 8):
	
	`etype=Disconnection`**,** `loop=0:`
	disconnessione di zona o generale (`zone=0`); nel secondo caso, descrizione unica.
	
	`etpe=Alarm, loop=0:`
	Attivazione manuale sirene, descrizione unica.

zone	Numero di **zona** dell'elemento. Casi particolari (rif. pag. 8): `etype=Disconnection`**,** `loop=0, zone=0:` disconnessione generale, descrizione unica.
address	**Indirizzo** dell'elemento.
timestamp	**Data e ora** dell'evento, come scritte dalla **centralina** con il proprio orologio. Nel caso la data contenuta nel pacchetto non fosse valida viene indicato 1 Gennaio 0001, 00:00:00.
desc1	**Prima descrizione** fornita dalla centralina (fino a 40 byte); usare `GetDesc1()` per leggerla correttamente come stringa. Descrive la zona, salvo casi particolari nei quali vi è una sola descrizione (rif. pagg. 7,8).
desc2	**Seconda descrizione** fornita dalla centralina (fino a 40 byte); usare `GetDesc2()` per leggerla correttamente come stringa. Descrive il 'punto', salvo casi particolari nei quali vi è una sola descrizione (rif. pagg. 7,8).

value	Valore **dipendente dal tipo** di evento; corrisponde a "Dato / Cod. Averia" nella tabella a pag.7.
	
	In particolare:
	
	`etype=PreAlarm:`
	`value` contiene il valore analogico.
	
	`etype=Fault, value<100:`
	Guasto individuale, rif. pag.8
	
	`etype=Fault, value>=100:`
	Guasto generale, descrizione unica, rif. pag. 7.
	
	Nei due ultimi casi la libreria non fornisce le descrizioni indicate nel manuale, in quanto già presenti nei campi `desc1`, `desc2`.

eptype	Tipo del pacchetto esteso, corrispondente a "Tipo" nella tabella a pag.7. (non documentato)
GetEtypeDesc()	Ritorna una **stringa** corrispondente al **tipo** dell'evento, ad esempio `"Fault"` per un evento con `etype=Fault`.
GetDesc1(encoding)	Ritorna la **prima descrizione** sotto forma di stringa, spazi inclusi, usando l'encoding specificato. Vedi `desc1` sopra.
GetDesc2(encoding)	Ritorna la **seconda descrizione** sotto forma di stringa, spazi inclusi, usando l'encoding specificato. Vedi `desc2` sopra.
Describe(encoding)	Ritorna una stringa con una completa **descrizione leggibile** dell'evento, comprendente tipo, timestamp, valori e descrizioni.

6.3 classe Packet

(note)	Rappresenta un **pacchetto dati** ricevuto dalla centralina (una sequenza di `byte`). È accessibile a scopo di debugging. Alcuni metodi particolari non sono documentati e non vanno usati.
`Packet()`	**Crea un pacchetto dati** con il buffer alla massima dimensione, inizializzato a 0. Nell'uso normale **non è necessario** creare pacchetti; questo costruttore è usabile per debugging e simulazioni.
`buffer`	Il pacchetto come array di **`byte`**. Nel caso dei pacchetti contenuti nel log, ha la minima lunghezza necessaria per contenere i dati.
`Describe(encoding, showRaw)`	Ritorna una stringa con una completa **descrizione leggibile** del pacchetto. Se `showRaw=true` la stringa è composta da due linee, la prima delle quali contiene il byte dump del pacchetto. A differenza di `Event.Describe()` sono descritti anche i tipi di pacchetto che non generano eventi. Le due **descrizioni** contenute nei dati sono presentate separatamente, senza interpretare logicamente il pacchetto.
`ToString()`	Ritorna una stringa con il **byte dump** del pacchetto, con separatori.

7 Versioni di questo documento

v 1.0 29 Luglio 2013	Prima versione.