Piattaforma Applicativa Gestionale

Modulo Communication Framework

Release 9.2

COPYRIGHT 1992 - 2016 by **ZUCCHETTI S.p.A.**
Tutti i diritti sono riservati. Questa pubblicazione contiene informazioni protette da copyright. Nessuna parte di questa pubblicazione può essere riprodotta, trascritta o copiata senza il permesso dell'autore.

TRADEMARKS
Tutti i marchi di fabbrica sono di proprietà dei rispettivi detentori e vengono riconosciuti in questa pubblicazione.

Pagina lasciata intenzionalmente vuota.

Indice


Note

1 Rilevazione Dati

- DISPOSITIVI DI RILEVAZIONE DATI
- SPECIFICHE FUNZIONALI

Introduzione

Il modulo di Communication Framework di Ad Hoc è stato sviluppato nell'ottica di conseguire due scopi:
- da una parte si intende fornire allo sviluppatore di applicazioni verticali uno strato software teso ad interfacciare i dispositivi di rilevazione dati prodotti da Zucchetti TMC.
Questo strato incapsula i dettagli di basso livello svincolando lo sviluppatore dalla conoscenza del firmware dei diversi modelli di terminale, permettendogli così di concentrarsi esclusivamente sull'applicazione vera e propria.
- dall'altro lato il Communication Framework fornisce all'utente finale un'insieme di anagrafiche per la definizione delle caratteristiche della rete; inoltre, a tempo di esecuzione, testa lo stato dei terminali evidenziando eventuali anomalie di funzionamento e coordina il funzionamento dei dispositivi, gestendo in modo multitasking reti formate da terminali di diverso tipo, operanti con modalità diverse e con diverse funzionalità.

Dunque, poiché di fatto il Communication Framework rappresenta sia uno strumento software che viene messo a disposizione degli sviluppatori sia un modulo destinato agli utenti finali, la struttura del presente manuale sarà suddivisa in due parti principali: un prima parte avrà lo scopo di introdurre le problematiche relative alla rilevazione automatica dei dati e di inquadrare in questo contesto il Communication Framework: in particolare verranno descritte le caratteristiche fondamentali del modulo con particolare riferimento allo sviluppo di applicazioni di interfaccia tra i dispositivi Zucchetti TMC e Ad Hoc; la seconda parte invece descriverà i dettagli relativi alle anagrafiche e al funzionamento del Communication Framework a run-time.

DISPOSITIVI DI RILEVAZIONE DATI

Trattando di rilevazione automatica dati la prima cosa che vale la pena notare è che ci si trova di fronte ad una tecnologia di tipo orizzontale, nel senso che molteplici possono essere i contesti in cui essa può essere impiegata.
I dispositivi di rilevazione automatica infatti possono essere utilizzati in applicazioni che vanno dalla rilevazione degli accessi alla rilevazione delle presenze e ancora alla logistica e alla gestione della produzione.
In tutti questi contesti numerosi sono i vantaggi che possono essere tratti dall'impiego di dispositivi di questo tipo: tra tutti ricordiamo la possibilità di poter disporre e trattare informazioni in tempo reale; la possibilità di gestire transazioni in maniera decentrata; la riduzione dei tempi e dell'uso di risorse (e dunque dei costi) nel processo produttivo; la riduzione della possibilità di errore nel trattamento dell'informazione.
Al fine di rendere ulteriormente visibili tali vantaggi proponiamo di seguito alcuni esempi di possibili applicazioni, suddivisi in tre grandi contesti: la logistica e il magazzino, la gestione della produzione, il controllo della qualità.

Logistica / Magazzino

Un primo esempio di applicazione consiste nella possibilità di utilizzare all'interno di un magazzino una o più postazioni che permettano di interrogare la base di dati riguardo agli articoli.
Ogni terminale può implementare uno o più tipi di interrogazione: potrà dunque richiedere il prezzo in base ad un certo listino, oppure richiedere la giacenza in un certo magazzino o ancora effettuare l'interrogazione dell'ordinato.
Il vantaggio in questo caso consiste nel poter disporre di queste informazioni direttamente sul campo ed in tempo reale.

Un driver di esempio di questo tipo viene fornito a corredo del Communication Framework e verrà discusso dettagliatamente in Appendice

Ancora, è possibile pensare di predisporre un sistema di dogane per gestire in maniera decentrata le movimentazioni di magazzino: applicando un'etichetta con un codice a barre su ogni imballo, i carichi e gli scarichi di articoli e materiali potranno essere registrati automaticamente in modalità on line.
Il vantaggio consisterà principalmente nella possibilità di gestire in modo decentrato tutte le transazioni, senza dover aggiornare i dati da un unico posto di lavoro: questo diminuirà in maniera consistente i tempi e contemporaneamente permetterà di aumentare l'accuratezza nel trattamento dell'informazione.
Infine si può pensare anche ad applicazioni di tipo "kanban elettronico".
Il kanban è un concetto legato alle logiche di tipo just-in-time: esso prevede che in corrispondenza di una qualsiasi situazione di scarico si verifichi una qualche reazione che generi automaticamente un fabbisogno.
In origine questo tipo di procedura era manuale: "kanban" infatti è una parola giapponese che significa "cartellino" ed indicava un cartellino fisico attaccato ai colli.

Una volta che questi venivano movimentati il cartellino veniva staccato: un cartellino staccato aveva il significato di un ordine di produzione.
Questa procedura può essere automatizzata mediante l'uso combinato di terminali per la rilevazione dati e di un applicativo gestionale quale Ad Hoc: in corrispondenza di uno scarico da magazzino, un cartellino elettronico viene rilevato automaticamente per mezzo di un terminale e i dati vengono imputati direttamente sul gestionale, generando da una parte un ordine di lavorazione e dall'altra un ordine a fornitore.
Addirittura si può pensare di estendere ulteriormente la cosa, gestendo in maniera trasparente la supply-chain, ovvero inoltrando sempre automaticamente l'ordine al fornitore via e-mail.

Gestione della Produzione

Un primo esempio può essere quello dell'avanzamento automatico delle dichiarazioni di produzione: quanto a questo tipo di applicazione vale la pena soffermarsi sulla differenza che esiste nella procedura che deve essere seguita confrontando un contesto in cui non si utilizzino sistemi di rilevazione automatica dati ed un contesto in cui si faccia un uso combinato dei terminali Zucchetti TMC e di Ad Hoc.
Nel primo caso una volta stampato l'ODL, questo dovrà essere passato all'operatore in produzione il quale dovrà annotare – a mano – una quantità di dati: la propria matricola, l'orario di inizio lavoro, l'orario di fine lavoro, la quantità prodotta, la risorsa utilizzata, le eventuali ore di perdita con le relative causali.
Una volta compilata, la dichiarazione di produzione dovrà essere passata all'indietro ad un altro operatore che avrà il compito di imputare i dati sul gestionale.
Utilizzando invece in modo combinato le tecnologie Zucchetti TMC e Ad Hoc la situazione sarà la seguente: l'ODL verrà stampato con un codice a barre e verrà passato all'operatore in produzione; questi dovrà strisciare il proprio badge sul terminale, leggere il codice a barre dell'ODL/fase e dichiarare quantità prodotta ed eventuali ore di perdita con le relative causali.
A questo punto i dati saranno immediatamente disponibili in Ad Hoc senza bisogno di ulteriori azioni o passaggi di documentazione cartacea.
Non solo: si può pensare di estendere la procedura generando automaticamente i carichi e gli scarichi dei magazzini prodotti finiti e work-in-process e lanciando la relativa stampa dei documenti di accompagnamento materiali.
Ancora, l'imputazione delle ore di perdita con le relative causali potrà portare a stilare automaticamente un serie di statistiche sulle ore di inefficienza che potranno fornire la base per un miglioramento del processo produttivo.

Un driver di esempio di questo tipo viene fornito a corredo del Communication Framework e verrà discusso dettagliatamente in Appendice

Un'altra applicazione classica è quella della rilevazione presenze: i dispositivi di rilevazione dati potranno essere utilizzati per consuntivare automaticamente le ore lavorate da un certo operatore, con la possibilità di associarvi ogni volta una causale.
Questo tipo di applicazioni trova impiego in numerosi contesti produttivi, primo fra tutti quello, importantissimo, delle produzioni a commessa: le ore lavorate infatti potranno in quel caso essere associate automaticamente ad una specifica commessa e ad una specifica attività.
Questo tipo di applicazione risulta utile poi anche in casi in cui si faccia pesante ricorso a ditte esterne: non occorre sottolineare l'importanza di questa situazione, vista la valenza che ha assunto negli ultimi anni il ricorso all'outsourcing in diversi contesti produttivi.

Controllo della Qualità

Anche per quanto riguarda il controllo della qualità l'utilizzo dei dispositivi di rilevazione dati può significare un netto miglioramento delle prestazioni in molte delle procedure previste.
In questo contesto i terminali possono essere impiegati durante tutto il processo produttivo.
Inizialmente possono essere utilizzati all'arrivo dei materiali per la gestione dell'accettazione: i materiali provenienti da produttori certificati verranno automaticamente riconosciuti come materiali di tipo *free pass*, mentre gli altri verranno sottoposti alle opportune verifiche di conformità.
Al solito verranno generate automaticamente le relative transazioni di magazzino e i dati raccolti andranno ad alimentare la base di dati utilizzata per effettuare le statistiche.
In seguito i terminali potranno essere utilizzati per la rilevazione automatica delle causali di scarto direttamente dal reparto competente fornendo i dati per le analisi PPM (Parti Per Milione) sul processo produttivo e per i controlli di qualità finali che porteranno alla valutazione della percentuale di difettosità del processo.

SPECIFICHE FUNZIONALI

Poiché, come si è visto, molte e diverse sono le modalità di impiego delle tecnologie di rilevazione automatica, dal punto di vista del software risulta chiaramente impossibile scrivere "l'Applicazione" che si adatti a tutte le esigenze dei diversi contesti produttivi.
Analogamente risulta poco praticabile la via di costruire un insieme di applicazioni base da cui ricavare le applicazioni verticali destinate agli utenti finali: questo perché l'interazione con il firmware dei terminali è talmente stretta che un tale approccio significherebbe riscrivere ogni volta le applicazioni praticamente ex novo, confrontandosi ogni volta con le diverse configurazioni della rete di terminali da gestire, segnatamente quindi con i diversi modelli dei terminali, i diversi tipi di concentratori, i diversi protocolli di rete, ecc.

L'idea che sta alla base del Communication Framework è stata quindi quella di offrire uno strumento che permetta agli sviluppatori di poter realizzare le applicazioni verticali in modo semplice e veloce, svincolandosi dalla conoscenza di tutti i dettagli di basso livello, previa un sola acquisizione di know how che, tra le altre cose, ha il vantaggio – non trascurabile – di aumentare il valore della loro conoscenza delle possibilità offerte da CodePAINTER.

La figura seguente illustra la situazione architetturale standard in cui dovrebbe essere utilizzato il Communication Framework:

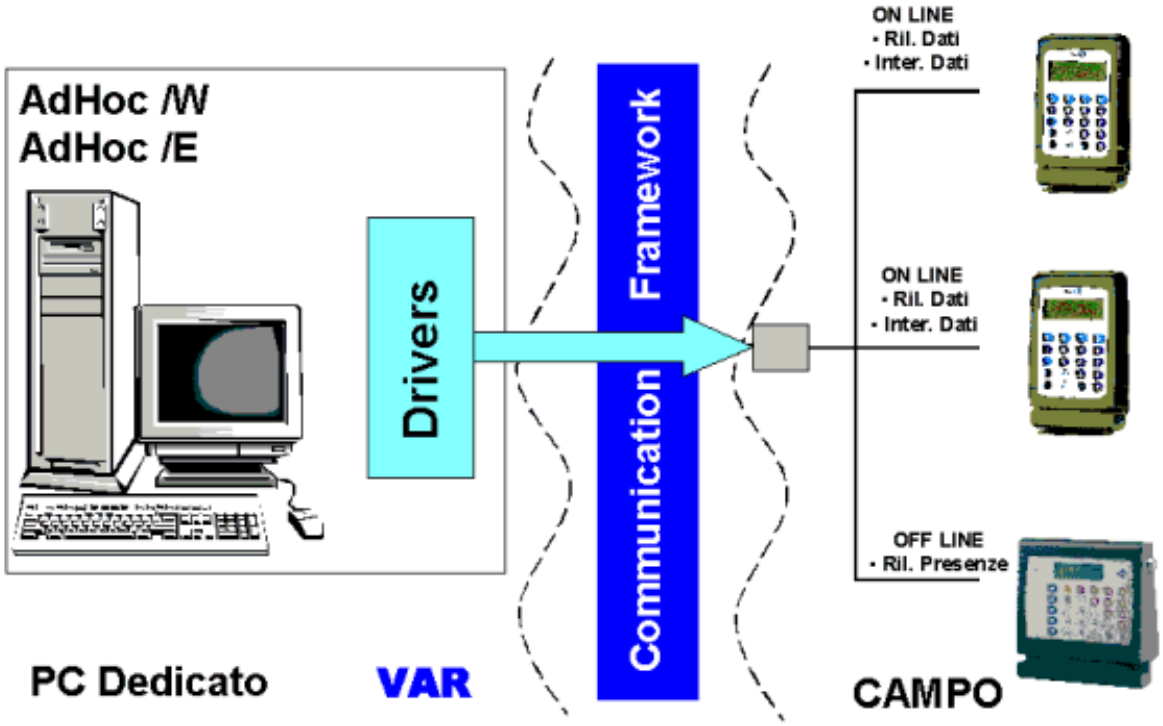


Fig. 1.1 – Reti Zucchetti TMC con Communication Framework

Nella situazione generica una rete di terminali TMC viene collegata ad una workstation host su cui sia stato installato Ad Hoc, con il modulo di Communication Framework attivo, che funge da server di rete. I terminali possono essere di modelli diversi (TRAX/2, TRAX/2P o PROX) e sono collegati per mezzo di un opportuno concentratore di rete (MICROLD o PSLD) ad una delle porte seriali della workstation.

Qui e nel seguito: per i dettagli relativi ai collegamenti e in generale alle caratteristiche hardware si rimanda alla manualistica Zucchetti TMC, disponibile anche all'URL: http://www.tmc-srl.it

I terminali possono lavorare sia in modalità on-line che in modalità off-line e possono adempiere ciascuno ad una diversa funzionalità.
E' fortemente consigliato che il server di rete sia costituito da un PC dedicato, soprattutto in riferimento alle applicazioni di tipo on-line, dove risulta essere critica la tempestività nei tempi di risposta.

Il Communication Framework rappresenta l'interfaccia che mette a disposizione dello sviluppatore un insieme di primitive di alto livello per la scrittura dei driver di funzionalità, incapsulando tutti i dettagli di basso livello legati al firmware dei terminali: allo sviluppatore viene tolta l'incombenza di dover coordinare a tempo di esecuzione il funzionamento dei diversi dispositivi presenti sulla sulla rete; di dover evidenziare e gestire gli eventuali malfunzionamenti che possono verificarsi; di dover gestire i diversi tipi di timeout che vengono impostati.

Per quanto riguarda il coordinamento dei terminali presenti sulla rete, il Communication Framework implementa un sistema di tipo multitasking che realizza una politica di tipo round-robin: questo significa che a tempo di esecuzione il Communication Framework garantisce che tutti i terminali presenti sulla rete vengano periodicamente serviti (con una frequenza che dipende dal parametro *clock timer*, definito a livello di anagrafica Server Net TMC); i terminali vengono serviti secondo una gestione di tipo circolare: dopo aver eseguito un passo dell'applicazione (corrispondente alle operazioni contenute in uno stato dell'automa) il controllo viene ceduto nuovamente al Communication Framework che interpellerà il terminale successivo. Se un terminale non risponde, l'anomalia verrà immediatamente segnalata a video dal Communication Framework e il terminale non verrà più interpellato fino a quando non sarà stato esplicitamente riattivato.

Il Communication Framework permette poi di impostare diversi tipi di timeout sulla rete dei terminali: il significato dei diversi tipi di timeout e l'uso che dovrà esserne fatto verrà discusso dettagliatamente in seguito.
In questa sede vogliamo invece sottolineare come tutti i timeout vengano gestiti in maniera totalmente automatica (e quindi trasparente allo sviluppatore) dal Communication Framework, che si occupa di attivare i timeout e di gestire il comportamento dell'applicazione qualora uno di essi scada.

Le funzionalità che dovranno essere sviluppate per costruire le applicazioni verticali saranno implementate per mezzo di due elementi:

- un file di codice Visual FoxPro con estensione *.prg* che contiene l'estensione della classe base dei terminali necessaria per l'applicazione che si sta sviluppando;
- un driver corrispondente ad un batch CodePAINTER, che rappresenta l'applicazione vera e propria.

Per quanto riguarda il primo di questi due elementi, il Communication Framework mette a disposizione una classe base di terminali: questa classe viene definita per mezzo di un insieme di proprietà e metodi che permettono di implementare l'insieme delle funzionalità di base che devono essere associate ad un terminale generico.
L'utilizzo di tali proprietà e metodi viene incapsulato per mezzo di un insieme di primitive di alto livello che funge da interfaccia nei confronti degli oggetti di tipo terminale.
Chiaramente però questa classe base non può essere sufficiente a soddisfare le esigenze delle diverse applicazioni verticali.
Il Communication Framework è stato disegnato in modo da sfruttare appieno le potenzialità offerte dalla tecnologia di programmazione ad oggetti: questo permette di ottenere una completa corrispondenza tra terminali fisici e terminali logici, estendendo facilmente la classe base dei terminali per mezzo del meccanismo dell'ereditarietà, in modo da disegnare di volta in volta una classe di terminali che si adatti perfettamente alle necessità dell'applicazione che deve essere sviluppata.
Questo viene fatto inserendo l'estensione della classe in un file con estensione *.prg* che deve essere inserito in una delle cartelle in *path*.
Descriviamo ora lo schema per mezzo del quale devono essere costruite le applicazioni vere e proprie: tali applicazioni saranno essenzialmente dei driver corrispondenti a dei batch CodePAINTER.
Di fatto i driver che dovranno essere realizzati devono implementare un automa a stati finiti: formalmente un automa viene definito da un insieme finito di stati in cui può trovarsi l'applicazione e da un insieme di transizioni tra essi.
Più precisamente si ha una transizione di stato quando una automa che si trovi in un certo stato passa in un altro stato in conseguenza del verificarsi di un certo evento (ad es. la ricezione di un messaggio).

Lo schema standard di un driver si compone di 3 parti principali:

- **dichiarazioni**: in questa parte vengono effettuate le dichiarazioni di variabile. In particolare deve essere dichiarato l'oggetto che rappresenta il terminale su cui deve essere eseguita l'applicazione implementata dal driver

- **gestione asincrona dei tasti funzionali**: in questa parte viene inserito un comando *case* che permette di gestire i tasti funzionali in maniera asincrona rispetto allo stato del terminale: questo significa che una funzionalità associata ad un certo tasto e definita in questa sezione sarà valida in qualsiasi punto del driver, ovvero in qualsiasi stato in cui si trovi il terminale. Questo può essere utile ad esempio per gestire una funzione di terminazione della transazione in corso.

Per una tabella di valori che permettono di gestire i diversi tasti funzionali si rimanda all'appendice

- **gestione automa dell'applicazione**: come abbiamo avuto già modo di evidenziare, una applicazione verticale tesa ad interfacciare i dispositivi Zucchetti TMC deve implementare un automa a stati finiti. Questa sezione, pertanto, dovrà essere strutturata come un *case*, in cui ad ogni ramo corrisponda uno stato dell'applicazione. Lo stato dell'applicazione viene identificato per mezzo della proprietà nSTATO dell'oggetto terminale chiamante e le transizioni di stato vengono effettuate per mezzo di assegnamenti espliciti: nSTATO=...

Il Communication Framework inoltre mette a disposizione dello sviluppatore un insieme di primitive di alto livello che permettono di implementare facilmente la comunicazione con i terminali.

Per uno schema dettagliato per la costruzione di un driver di funzionalità e per gli strumenti messi a disposizione allo scopo si rimanda all'appendice


Note

2 Archivi e Funzionalità

- SERVER NET TMC
- DRIVER
- TERMINALI
- MONITOR

Introduzione

In questo capitolo vengono presentate le anagrafiche e le funzionalità offerte dal Communication Framework. Il modulo prevede tre anagrafiche:

- Server di rete TMC
- Driver di Funzionalità
- Terminali

Oltre alle anagrafiche è presente una procedura Monitor che implementa il nucleo fondamentale delle funzionalità offerte dal Communication Framework

La figura seguente mostra il Menu relativo alla gestione del Communication Framework; ogni voce verrà dettagliatamente trattata nel rispettivo paragrafo.

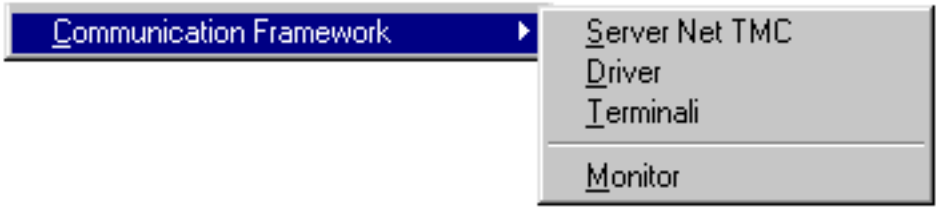


Fig. 2.1 – Menù Communication Framework

SERVER NET TMC

Tramite questa gestione vengono archiviate le informazioni relative alle workstation che devono fungere da server per le reti di dispositivi di rilevazione dati Zucchetti TMC.
I server di rete vengono definiti in base ad un insieme di informazioni che permettono di identificare univocamente la workstation e di descrivere in che modo la rete di terminali sia ad essa collegata.
La gestione dei server di rete TMC presenta due tab, uno per le IMPOSTAZIONI BASE ed uno per le OPZIONI. Di seguito viene fornito il dettaglio della gestione.

Ricerca

Può essere effettuata mediante l'unica chiave:
- Nome Server

Altrimenti è possibile accedere alla cartella Elenco, selezionando la chiave di ricerca che interessa, ad esempio cliccando sulla colonna Descrizione.

Carica

Di seguito viene presentato l'elenco dei campi richiesti ed il loro significato

Impostazione di base

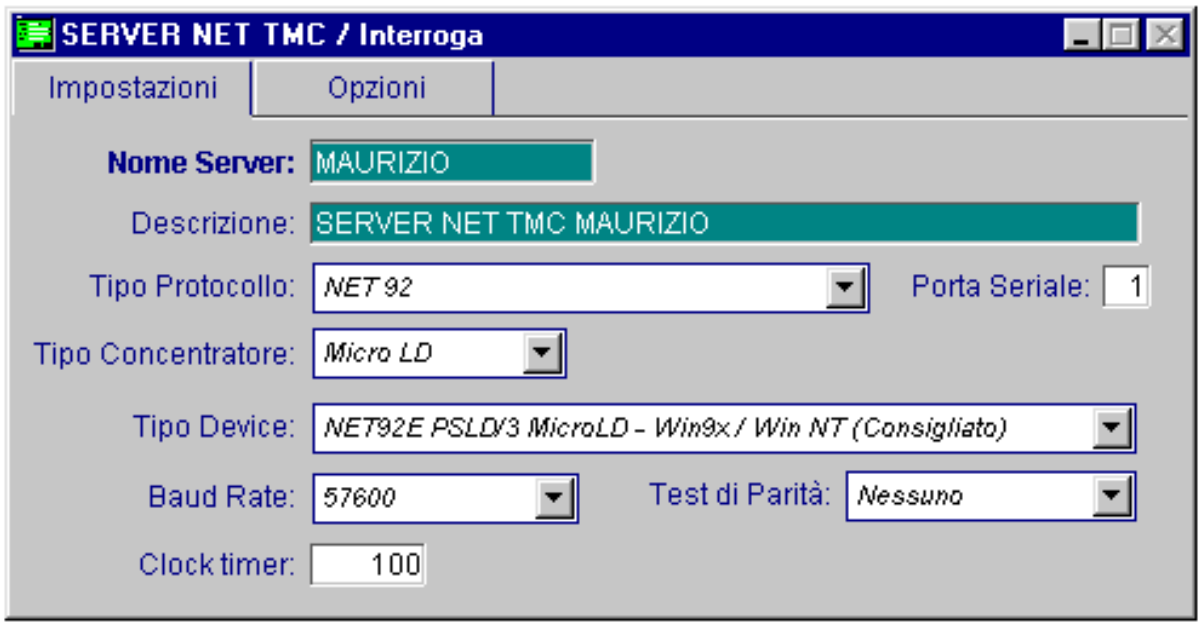


Fig. 2.2 – Server di Rete TMC – Impostazioni Base

Nome Server

La procedura richiede come primo campo il codice alfanumerico che permette di identificare la workstation.
Tale codice in ambienti di rete Windows deve corrispondere al codice di identificazione utilizzato dalla rete per la workstation.

Descrizione

Una descrizione mnemonica della workstation (ad es. il nome dell'utente preposto a quella postazione di lavoro)

Tipo Protocollo

Il tipo di protocollo di rete utilizzato; la combo box permette di impostare:
- NET 92

Per il momento il supporto per le reti Ethernet e per i collegamenti seriali RS232 non è ancora disponibile

Porta Seriale

Il numero (1, ... , 4) della porta seriale cui è collegata fisicamente la rete di terminali.

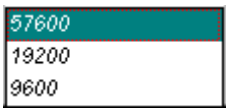
Tipo Concentratore

Lo specifico dispositivo Zucchetti TMC utilizzato per collegare la rete di terminali alla workstation server. La combo box permette di impostare questo parametro a:
- *MICRO LD*: concentratori per reti di piccole dimensioni (fino a 5 terminali)
- *PSLD*: concentratori per reti di grandi dimensioni (fino a 254 terminali)

Per il momento il supporto per i concentratori di tipo Current Loop non è ancora disponibile

Tipo Device

Parametro per la configurazione della connessione di rete sulla base del tipo di protocollo, tipo di concentratore e sistema operativo utilizzati. La combo box offre le seguenti possibilità di scelta:
- NET920 old PSLD - microLD mode 1/2 & winNT (RTS mode 0)
- NET92N PSLD/2-PSLD/3-microLD mode 2 & microLD mode 1/2 /winNT (RTS mode 1)
- NET92E PSLD/3 - microLD mode 2 & win9x - PSLD/3 & winNT

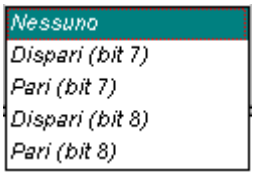


Baud Rate

La velocità di comunicazione espressa in baud. La combo box è editabile solo nel caso di reti basate su protocollo NET92 ed offre le seguenti possibilità di scelta:

Si noti che negli altri casi la velocità di comunicazione viene settata automaticamente dalla procedura

Test di Parità



Controllo che permette di impostare il tipo di test di parità da effettuare e la lunghezza della parola. Il controllo non è editabile nel caso di connessioni di tipo Ethernet (gestione automatica da parte della procedura) ed offre le seguenti scelte:

Clock Timer

L'intervallo (espresso in millisecondi) con cui viene effettuato un ciclo di attività sulla rete; all'interno del Communication Framework è presente un timer: ogni volta che il timer scatta il sistema effettua un ciclo per servire i terminali. Il *clock timer* definisce l'intervallo con cui far scattare il timer.

Intervalli troppo contenuti, specialmente nel caso di reti di piccole dimensioni e con server veloce, possono comportare in alcuni casi una difficile interazione da parte dei dispositivi di input dei terminali. Dai test effettuati un valore accettabile dalla maggior parte delle configurazioni utilizzate si aggira intorno ai 100 ms

Opzioni

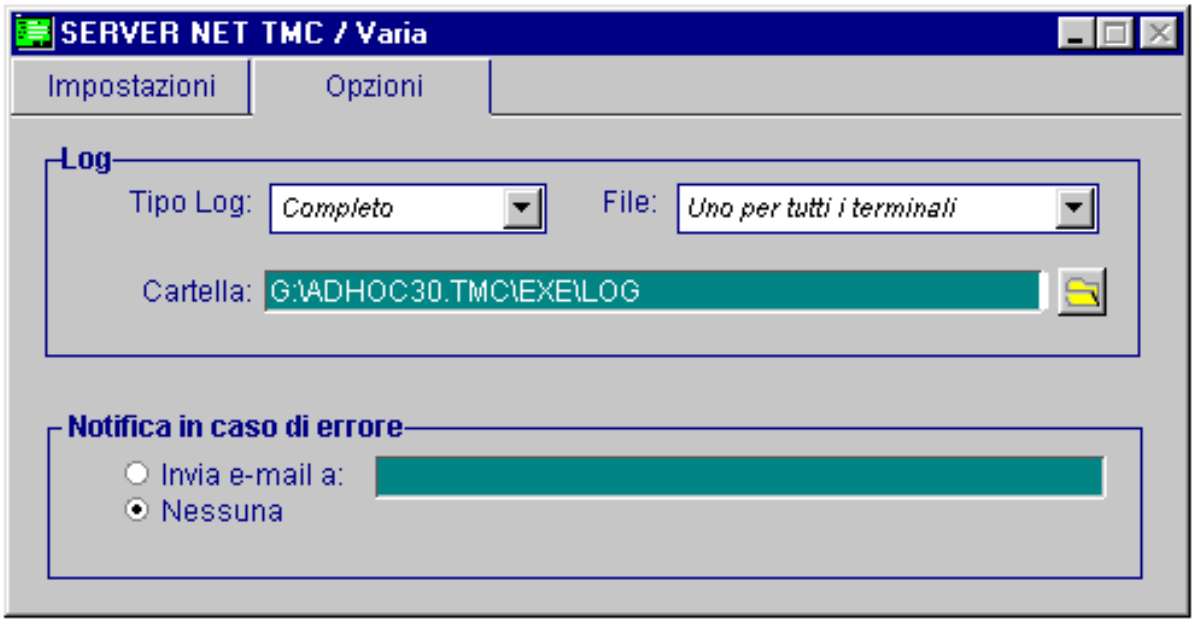
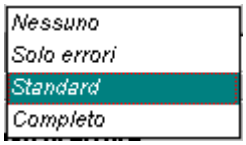


Fig. 2.3 – Server di Rete TMC - Opzioni

Tipo Log



La politica di logging che si intende utilizzare: le scelte offerte dalla combo box sono le seguenti:

- *Nessuno*: nessun messaggio di log viene registrato;
- *Solo errori*: vengono registrate solo le segnalazioni di errore o di fallimento, che possono pregiudicare il funzionamento di un terminale;
- *Standard*: vengono registrate anche le segnalazioni di avviso ("*warning*") relative a possibili anomalie, che però non pregiudicano il funzionamento del terminale, e la messaggistica relativa ad eventi particolarmente importanti;
- *Completo*: viene registrata tutta l'attività in corso sul terminale, ivi incluse le operazioni di trasmissione e ricezione di messaggi.

L'attivazione del logging completo deve essere effettuata solo nel caso in cui si voglia verificare il funzionamento di una applicazione a fini di debugging: in alcuni casi infatti i file di log possono raggiungere in breve tempo dimensioni considerevoli

File

La procedura genera ogni giorno nuovi file di log. Questo controllo permette di specificare come distribuire il log su tali file; la combo box permette di scegliere tra:

- *Uno per ogni terminale*: viene generato un solo file di log in cui vengono registrate le segnalazioni di tutti i terminali con log attivo
- *Uno per ciascun terminale*: viene generato un file di log per ogni terminale su cui il log è attivato

Cartella

Permette di specificare il percorso della cartella in cui salvare i file di log; utilizzando il pulsante a lato è possibile effettuare una ricerca visuale.

☑ Notifica in caso di errore

Permette di specificare se notificare automaticamente una situazione di fallimento: se nel radio button viene impostato il valore "*Invia e-mail a*", verrà inviato un messaggio all'indirizzo di posta elettronica specificato

DRIVER

Tramite questa gestione vengono archiviate le informazioni relative ai driver delle funzionalità (ovvero alle applicazioni verticali) da associare a ciascun terminale.
I driver vengono definiti sulla base di un batch CodePAINTER, di un file *.prg* che contiene l'estensione dello stato del terminale per l'applicazione corrente e di un insieme di dati per la definizione dei diversi tipi di timeout.
Vediamo come si presenta la gestione dei driver delle funzionalità:

Ricerca

Può essere effettuata mediante l'unica chiave:
- Codice Driver

Altrimenti è possibile accedere alla cartella Elenco, selezionando la chiave di ricerca che interessa, ad esempio cliccando sulla colonna Descrizione.

Carica

L'anagrafica presenta due tab: una pagina di Impostazioni Base ed una di Offline che permette di pianificare lo scarico dei dati per le applicazioni di tipo off line. Ecco di seguito l'elenco dei campi richiesti ed il loro significato.

Impostazione base

Fig. 2.4 – Driver – Impostazioni Base

Codice

Un codice alfanumerico per l'identificazione del driver

Descrizione

Una descrizione mnemonica del driver (ad es. il tipo di funzionalità implementata)

Applicazione

Il batch CodePAINTER che implementa il programma di gestione del terminale, di fatto l'applicazione verticale che si vuole realizzare

Estensione Terminale

L'identificazione del file .prg che contiene la ridefinizione, per estensione, della classe base dei terminali: questo permette di definire per ogni terminale uno stato che si adatti perfettamente alle esigenze dell'applicazione che si vuole realizzare

Modalità

Flag che permette di specificare se l'applicazione viene gestita in modalità on line oppure in modalità off line

Intervallo di Verifica

Questo campo ha un diverso significato a seconda che si stia implementando una applicazione di tipo on line o di tipo off line.

- Nel caso di applicazioni *on line* permette di specificare l'intervallo di tempo (espresso in secondi) allo scadere del quale una transazione in attesa deve essere considerata fallita, riportando il terminale nello stato iniziale. Una transazione viene considerata fallita quando il terminale rimane inattivo (ovvero non si registra attività sul terminale) per un periodo di tempo superiore all'intervallo di tempo impostato. Tale intervallo di tempo pertanto dovrà essere tarato in modo ragionevole sulla base della specifica applicazione che si intende realizzare;
- Nel caso di applicazioni *off line* permette di specificare ogni quanti secondi deve essere verificato il corretto funzionamento del terminale, in modo da prevenire malfunzionamenti al momento del polling. Nel caso di alcune applicazioni off line, infatti, il terminale può lavorare anche molto tempo senza bisogno di interagire con il PC server.

In entrambi i casi, impostando il parametro al valore 0 non verrà intrapresa alcuna azione di verifica.

Offline

Fig. 2.5 – Driver - OffLine

Controlli attivi solo in caso di applicazioni operanti in modalità *off line*: permettono di specificare in maniera dettagliata quando deve essere effettuato il polling di una applicazione off line implementata su una rete di terminali Zucchetti TMC di tipo programmabile (TRAX\2P, PROX). Il flag è rappresentato per mezzo di un radio button a due scelte:

Pianificazione Esecuzione

⊙ A Intervallo

viene proposto un campo numerico, entro cui si può inserire un'intervallo (espresso in secondi) allo scadere del quale viene effettuato il polling

⊙ A Scadenza

in questo caso è possibile impostare fino a 3 diversi orari giornalieri per il polling (campi ora/minuto identificati **con 1°,2°,3° Polling**); inoltre per mezzo di un radio button a tre scelte è possibile impostare la frequenza di polling, secondo le scelte seguenti:

Selezione

- ⊙ **Ogni Giorno**: attivando questa opzione il polling verrà effettuato ogni giorno, agli orari specificati
- ⊙ **Giorni Settimana**: attivando questa opzione si potrà editare una serie di sette check box corrispondenti ciascuno ad un giorno della settimana. Il polling verrà effettuato solo nei giorni (check box) che saranno selezionati, agli orari specificati
- ⊙ **Giorni Mese:** attivando questa opzione si renderà editabile una edit box in cui l'utente potrà specificare una serie di giorni del mese. Il campo provvede a suddividere automaticamente i giorni per mezzo di uno spazio bianco. Il polling verrà effettuato solo nei giorni del mese inseriti, agli orari specificati.

TERMINALI

Tramite questa gestione vengono archiviate le informazioni relative ai terminali Zucchetti TMC presenti sulla rete.
I terminali vengono descritti in termini di alcune caratteristiche hardware, della funzionalità che devono implementare e del server di rete cui sono collegati.
Vediamo come si presenta la gestione dei terminali:

Fig. 2.6 - Terminali

Ricerca

Può essere effettuata mediante l'unica chiave:

- Codice Terminale

Altrimenti è possibile accedere alla cartella Elenco, selezionando la chiave di ricerca che interessa, ad esempio cliccando sulla colonna Descrizione.

Carica

Ecco di seguito l'elenco dei campi richiesti ed il loro significato.

Codice

Un codice alfanumerico che identifica univocamente il terminale considerato

Descrizione

Una descrizione mnemonica del terminale (ad es. il luogo in cui è collocato)

☑ Abilitato

Flag che permette di abilitare / disabilitare un terminale (ad es. per permettere operazioni di manutenzione)

Applicazione

La funzionalità che deve essere svolta dal terminale

Server Host

Il server di rete TMC, ovvero la workstation host cui è collegato il terminale

Modello

Campo che permette di identificare a quale modello di terminale di rilevazione dati Zucchetti - - TMC si faccia riferimento. La combo box prevede le seguenti scelte:

- *TRAX*: terminale non programmabile
- TRAXP: terminale programmabile
- *PROX*: terminale programmabile di tipo evoluto

Indirizzo

L'indirizzo di rete a cui si trova il terminale.
Tale indirizzo deve corrispondere all'indirizzo memorizzato nel firmware del terminale. Chiaramente, assegnando lo stesso indirizzo a più terminali sulla solita rete si ottengono dei malfunzionamenti nel comportamento dei terminali: dovrà quindi essere cura di chi gestisce la rete di terminali assicurarsi che l'indirizzo memorizzato nel firmware del terminale corrisponda a quello memorizzato a livello di anagrafica

Indirizzo IP

L'indirizzo IP associato ad un terminale (utile nel caso di connessioni su rete Ethernet)

La gestione dell'indirizzo IP è strettamente legata alla gestioni delle reti Ethernet, il cui supporto per il momento non è ancora disponibile

Modem

Numero di telefono da associare al terminale nel caso di connessioni remote

Il supporto per la gestione dei terminali remoti per il momento non è ancora disponibile

Inizializzazione

Stringa che permette di inviare al terminale una sequenza custom di comandi da eseguire in fase di inizializzazione del terminale. La sintassi della sequenza di comandi deve essere basata sul macro-linguaggio di basso livello comprensibile dallo specifico modello di terminale.

Per maggiori approfondimenti su tali comandi si rimanda alla manualistica prodotta da Zucchetti TMC, disponibile anche all'URL: http://www.tmc-srl.it .

Opzioni

Propone una serie di quattro check box che permettono di specificare quali dispositivi di input esistono sul terminale

☑ Tastiera

☑ Lettore di Badge

☑ Barcode Interno

☑ Barcode Esterno

MONITOR

Questa funzionalità rappresenta il nucleo principale del Communication Framework.
Una volta che la rete di terminali Zucchetti TMC sia stata descritta in termini delle informazioni richieste dalle anagrafiche dei Server, dei Driver e dei Terminali, e una volta che siano stati effettuati i corretti collegamenti hardware, il Monitor può essere avviato.
I compiti cui presiede il Monitor sono numerosi e possono essere suddivisi entro due fasi distinte:
In fase di INIZIALIZZAZIONE il Monitor si occupa di reperire i terminali abilitati connessi alla workstation su cui viene lanciata la procedura; inoltre testa lo stato dei terminali evidenziando eventuali anomalie e malfunzionamenti sulla rete

In fase di NORMALE ATTIVITA' il Monitor si occupa di:

- coordinare il funzionamento dei diversi terminali presenti sulla rete;
- gestire i diversi tipi di timeout impostati sia a livello di anagrafica terminali che a livello di funzionalità;
- mantenere in modo online i terminali programmabili per i quali questo comportamento sia previsto dalla specifica funzionalità assegnata;
- gestire le cadute di alimentazione sulla rete dei terminali;
- attivare/disattivare una funzione di logging per monitorare il comportamento dei singoli terminali

Vediamo come si presenta l'interfaccia offerta dal Monitor una volta eseguita la fase di inizializzazione:

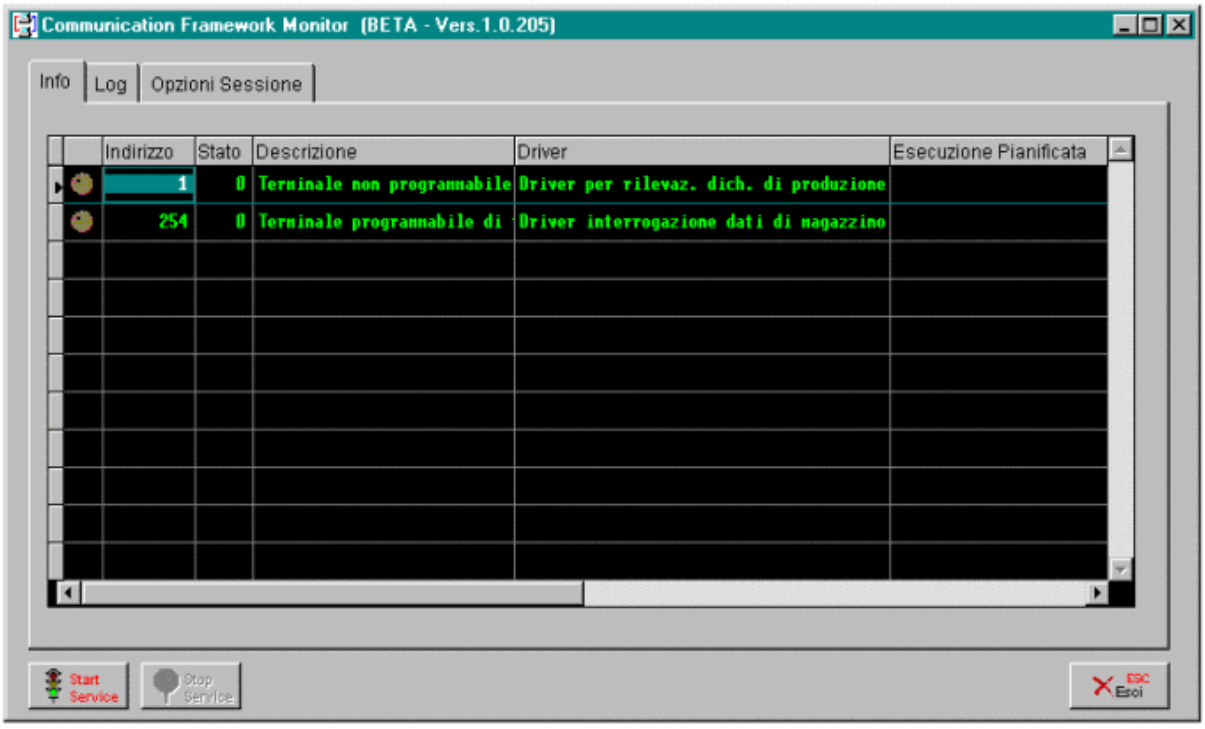


Fig. 2.7 – Monitor – Tab Info

Andiamo ad analizzare nel dettaglio i controlli presenti sulla finestra del Monitor.
L'interfaccia offre tre pulsanti, con le seguenti funzionalità:

Start Service

Permette di far partire il servizio: viene inizializzata la rete e vengono riscontrate eventuali anomalie sui terminali, dopodiché il Monitor inizia a servire i dispositivi in modo multitasking con una politica di tipo round-robin, secondo le diverse funzionalità associate a ciascuno di essi

Stop Service

Termina il servizio: le funzionalità sui terminali vengono bloccate (terminando eventuali transazioni attualmente in corso) e i dispositivi vengono resettati

Esci

Chiude il Monitor

La finestra presenta inoltre tre tab: un primo tab INFO è dedicato a rappresentare in modo immediato lo stato dei diversi terminali sulla rete; il secondo tab LOG è dedicato alla visualizzazione del logging sui terminali; infine il terzo tab OPZIONI SESSIONE permette di visualizzare e – in alcuni casi – modificare le opzioni relative al servizio in corso.

Info

In questa tab viene visualizzata una griglia a cinque colonne, in cui ad ogni riga corrisponde uno dei terminali reperiti sulla rete; il significato delle colonne è il seguente:
Nella prima colonna una bitmap indica lo stato di funzionamento del terminale: la bitmap può assumere quattro diversi colori:

- *Grigio* indica che il terminale non è attivo;
- *Verde* indica il normale funzionamento;
- *Rosso* indica che il terminale non risponde o che si è verificato un qualche tipo di anomalia;
- *Giallo* indica che il terminale potrebbe essere disconnesso o occupato: un motivo tipico potrebbe essere una caduta dell'alimentazione del terminale. La procedura cerca di riattivare automaticamente i terminali in questo stato: in caso di successo la transazione in corso viene ripresa a partire dall'ultimo stato consistente registrato;

Indirizzo:

colonna che indica l'indirizzo di rete a cui si trova il terminale

Stato:

indica lo stato dell'applicazione in cui si trova attualmente il terminale

Descrizione:

descrizione mnemonica del terminale

Driver:

descrizione mnemonica della funzionalità associata al terminale

Esecuzione Pianificata:

riporta l'istante della prossima esecuzione per le applicazioni di tipo offline

Inoltre cliccando con il pulsante destro del mouse su uno dei terminali viene presentato un menu di contesto con le seguenti opzioni:

- **Start**: fa ripartire il terminale selezionato;
- **Stop**: blocca il terminale selezionato;
- **Reload**: fa ripartire il terminale selezionato, caricando nuovamente le informazioni dalla base di dati. Questa funzionalità è utile nel caso in cui occorra variare alcune delle impostazioni registrate a livello di anagrafica;
- **Log on**: attiva la funzione di logging sul terminale selezionato;
- **Log off**: disattiva la funzione di logging sul terminale selezionato;

Log

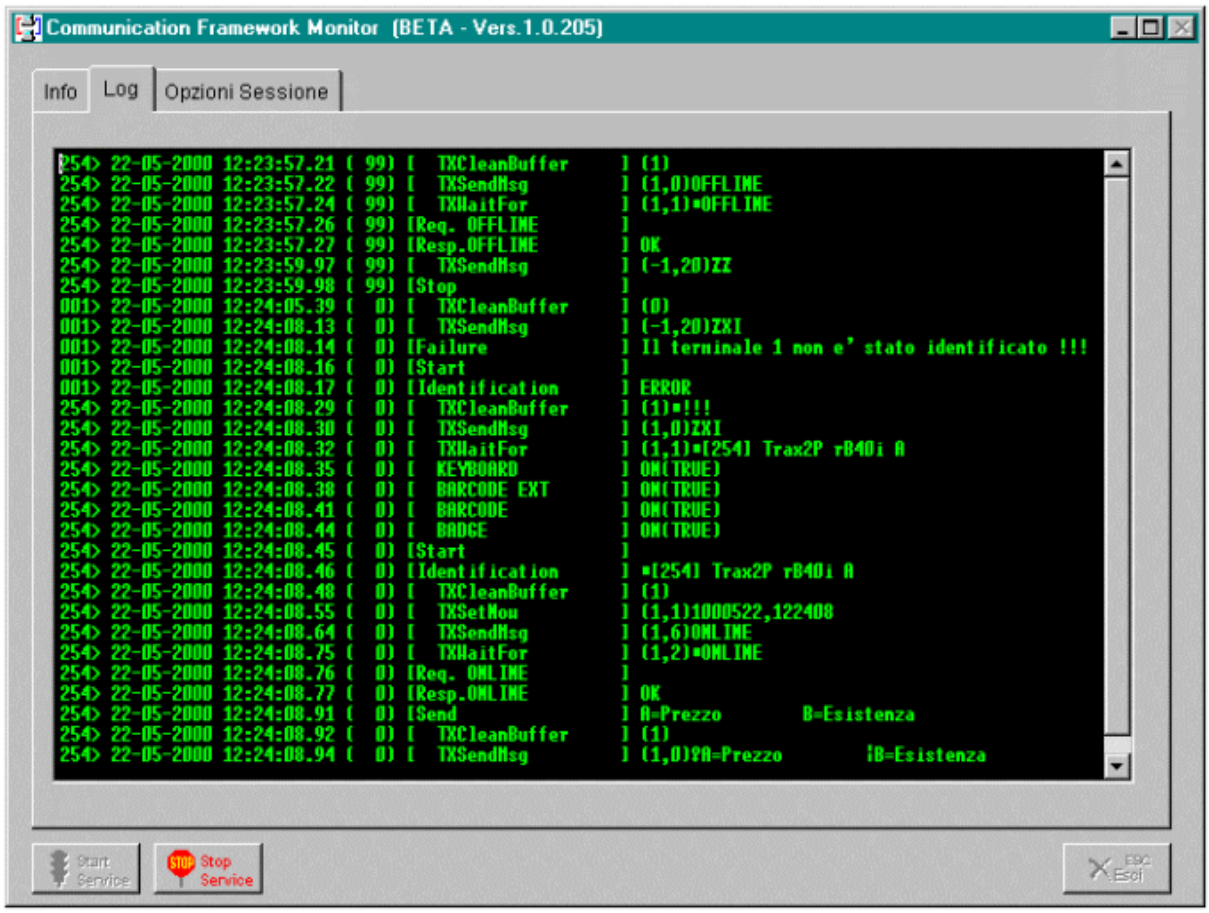


Fig. 2.8 – Monitor – Tab - Log

In questa tab è possibile visualizzare il tracciato dei log attivati sui diversi terminali. Per ogni riga di log riportata a video vengono evidenziati:

- l'indirizzo del terminale cui è relativa l'operazione che viene tracciata sul log
- l'ora dell'operazione
- lo stato in cui si trova il terminale
- il tipo di comando (send, receive, ecc.) od evento (failure, warning, ecc.) tracciato
- il messaggio inviato o ricevuto da terminale

Opzioni sessione

In questa tab è possibile visualizzare e modificare alcuni parametri, suddivisi in tre sezioni diverse, di cui di seguito viene fornito il dettaglio.

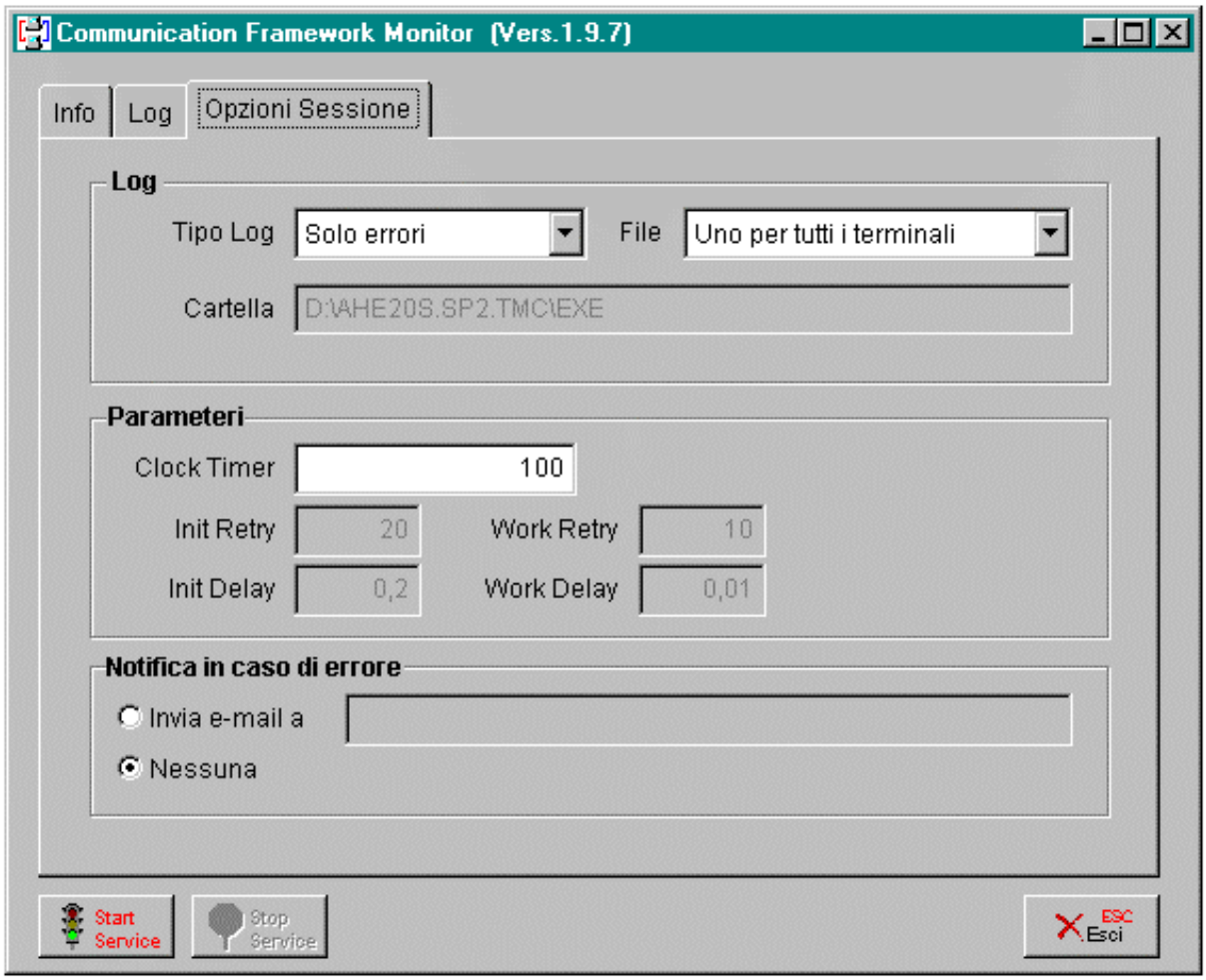


Fig. 2.9 – Monitor – Tab – Opzioni Sessione

Tipo Log

Permette di visualizzare / modificare il tipo della sessione di logging attualmente in corso. Le scelte possibili sono:

- *Nessuno*: nessun messaggio di log viene registrato;
- *Solo errori*: vengono registrate solo le segnalazioni di errore o fallimento, che possono pregiudicare il funzionamento di un terminale;
- *Standard*: vengono registrate anche le segnalazioni di avviso ("*warning*") relative a possibili anomalie, che però non pregiudicano il funzionamento del terminale, e la messaggistica relativa ad eventi particolarmente importanti;
- *Completo*: viene registrata tutta l'attività in corso sul terminale, ivi incluse le operazioni di trasmissione e ricezione di messaggi.

File

Permette di visualizzare / modificare come distribuire il log su file. La combo box permette di scegliere tra:

- *Uno per ogni terminale*: viene generato un solo file di log in cui vengono registrate le segnalazioni di tutti i terminali con log attivo
- *Uno per ciascun terminale*: viene generato un file di log per ogni terminale su cui il log è attivato

Cartella

Permette di visualizzare / specificare il percorso della cartella in cui salvare i file di log

Clock Timer

L'intervallo (espresso in millisecondi) con cui viene effettuato un ciclo di attività sulla rete.

Intervalli troppo contenuti, specialmente nel caso di reti di piccole dimensioni e con server veloce, possono comportare in alcuni casi una difficile interazione da parte dei dispositivi di input dei terminali. Dai test effettuati un valore accettabile dalla maggior parte delle configurazioni utilizzate si aggira intorno ai 100 ms

Init Delay

Visualizza il parametro di ritardo (in secondi) che la procedura utilizza in fase di inizializzazione, per assicurarsi che un terminale abbia avuto il tempo di effettuare una elaborazione

Init Retry

Visualizza il parametro relativo al numero di tentativi di comunicazione effettuati prima di segnalare un fallimento che la procedura utilizza in fase di inizializzazione

Work Delay

Visualizza il parametro di ritardo (in secondi) che la procedura utilizza in fase di normale attività, per assicurarsi che un terminale abbia avuto il tempo di effettuare una elaborazione

Work Retry

Visualizza il parametro relativo al numero di tentativi di comunicazione effettuati prima di segnalare un fallimento che la procedura utilizza in fase di normale attività

Invia e-mail a

Permette di specificare se notificare automaticamente una situazione di fallimento: se nel radio button viene impostato il valore "*Invia e-mail a*", verrà inviato un messaggio all'indirizzo di posta elettronica specificato


Note

3 Appendici

CLASSE BASE TERMINALI – PROPRIETÀ PUBBLICHE

Le proprietà della classe base dei terminali visibili allo sviluppatore delle applicazioni sono le seguenti:

nSTATO

Attributo numerico che indica lo stato in cui si trova il terminale ad un certo istante dell'elaborazione; viene utilizzato per gestire – tramite assegnamenti espliciti – le transazioni di stato dei terminali.

FUNCKEY

Attributo di tipo stringa che contiene l'indicazione dell'ultimo tasto funzionale premuto sul terminale.
La gestione dei tasti funzionali viene effettuata in modo trasparente dal Communication Framework, tuttavia questa proprietà non viene nascosta allo sviluppatore, per permettere la customizzazione delle funzionalità da associare a ciascun tasto.

Per una tabella dei valori che possono essere assunti da FUNCKEY si rimanda all'appendice Tabella di Riferimento Tasti Funzionali

CLASSE BASE TERMINALI – METODI PUBBLICI

In questa sezione viene fornito l'elenco delle primitive messe a disposizione degli sviluppatori:

Send(cMsg,cPrompt)

Primitiva che permette di inviare un messaggio *cMsg* ad un terminale, specificando in *cPrompt* una maschera di input (eventualmente vuota) secondo gli standard di programmazione Visual FoxPro che guidi l'immissione dei dati da terminale in accordo al formato voluto.
Ad esempio il prompt: "XXX" permetterà di leggere un dato alfanumerico di lunghezza 3, mentre il prompt "99" farà accettare solo dati numerici di lunghezza 2.
Si noti che il buffer dei terminali ha lunghezza 128, anche se la quantità di caratteri visualizzabili sul display è inferiore: questo significa che, anche se un prompt risulta troppo lungo per essere visualizzato completamente, tuttavia non ci saranno problemi per accogliere correttamente il dato.
La primitiva oltre all'invio vero e proprio del messaggio si occupa anche dei seguenti compiti:

- formatta opportunamente il messaggio secondo le caratteristiche del display del terminale;
- salva il contenuto della picture nello stato del terminale per futuri controlli di correttezza;
- attiva il timeout di verifica impostato a livello di anagrafica terminali;
- salva l'ultimo stato consistente dell'applicazione, in modo da poterlo ripristinare qualora se ne presentasse l'esigenza;
- gestisce eventuali malfunzionamenti nella comunicazione.

La funzione restituisce un valore numerico pari a 1 se la trasmissione è andata a buon fine, 0 altrimenti

Receive(@cBuffer)

Primitiva che permette di effettuare il polling del terminale e di salvare l'eventuale informazione ricevuta nella variabile *cBuffer.*
Più precisamente la primitiva restituisce 1 se il buffer di ricezione è pieno, 0 altrimenti. Eventuali malfunzionamenti sono gestiti in maniera trasparente.
La primitiva ha inoltre il compito di effettuare i controlli di correttezza sul formato del dato immesso da terminale, secondo la picture specificata nella send precedente: nel caso il dato non rispetti il formato voluto, un segnale acustico segnalerà l'errore e sarà ripristinato sul terminale l'ultimo stato consistente.

NextStateAfter (nState,nTimeout)

Primitiva che permette di impostare un timeout personalizzabile dallo sviluppatore dell'applicazione. Il significato della primitiva è che il terminale passerà nello stato *nState* dopo un numero di secondi pari a *nTimeout*.
Questa primitiva può essere utilizzata ad esempio per introdurre dei ritardi che permettano di

visualizzare temporaneamente sul display dei terminali il risultato dell'elaborazione: una volta ottenuto il dato da visualizzare si invia il dato al terminale e si fa partire il timeout con la *nextstateafter*, dopodichè si effettua una transizione di stato (anche fittizio).
Si noti che una volta impostato il ritardo, la gestione di questo timeout (e la conseguente transizione di stato) sarà delegata totalmente al Communication Framework e pertanto sarà trasparente allo sviluppatore dell'applicazione.

FunctionKeyNotManaged( )

Primitiva che permette di trattare i tasti funzionali non gestiti: viene applicata nei punti di programma in cui ci si attenda la pressione di un tasto funzionale sul terminale ed il tasto effettivamente premuto non corrisponda alle attese.
L'applicazione risponderà inviando un segnale acustico e ripristinando l'ultimo stato consistente sul terminale.

Beep(nTone)

Primitiva che permette l'emissione di un segnale acustico da parte del terminale: in particolare la variabile *tone* può assumere due valori corrispondenti ad un tono acuto (1) e ad un tono basso (0).

Keyboard(cCmd);

Primitiva che permette di operare con il dispositivo di input tastiera. Il significato della primitiva dipende dal valore del parametro *cCmd*:

- cCmd=ON: attiva il dispositivo di input, qualora questo sia presente. In caso contrario, se il log è attivo, verrà registrato un messaggio di warning
- *cCmd=OFF*: disattiva il dispositivo di input
- *cCmd=EXIST*: veifica l'esistenza del dispositivo di input (secondo l'informazione disponibile a livello di anagrafica terminali)

La primitiva restituisce un valore logico che indica nei primi due casi il successo o il fallimento del settaggio, mentre nel terzo caso indica l'esistenza o meno del dispositivo

Badge(cCmd)

Primitiva analoga alla KEYBOARD, relativa al lettore di badge magnetici

Barcode(cCmd)

Primitiva analoga alla KEYBOARD, relativa al lettore di barcode (penna o scanner) esterno

BarcodeInt(cCmd)

Primitiva analoga alla KEYBOARD, relativa al lettore di barcode interno

IsBadge (cBuffer,@cBadge,@cIO)

Primitiva che dato un codice *cBuffer* letto da terminale verifica se questo sia un badge e restituisce un valore logico opportuno: inoltre se la verifica è positiva, il codice del badge viene salvato in *cBadge* e il verso di entrata o uscita viene salvato in *cIO*

IORead (cCounter)

Primitiva che permette di leggere il valore di uno dei contatori interni ad un terminale, utilizzabili ad es. con funzioni di contapezzi.
I terminali dispongono di un certo numero di contatori minori e di un certo numero di contatori maggiori.
I contatori minori sono associati alle linee di input: ogni volta che un contatore minore ha eseguito un certo numero di scatti, memorizzato in un parametro nel firmware del terminale, viene generato un evento che permette di aggiornare il valore del corrispondente contatore maggiore.
Il numero di contatori validi varia a seconda del modello di terminale utilizzato: di seguito viene fornita una tabella dei valori validi per *cCounter* quanto ai contatori maggiori e ai contatori minori.
Inoltre vengono specificati i parametri di configurazione per i contatori minori da settare sul firmware dei terminali

MODELLO	*CONT. MAGGIORI*	*CONT. MINORI*	*Param. Configuraz. Cont. Min.*
TRAX		A	
TRAX/2P	A,B	A,B	36,37
PROX	A,B,C,D	A,B,C,D	36,37,38,39

Il significato del risultato restituito cambia a seconda del modello di terminale utilizzato: nel caso di terminali programmabili verrà restituito il valore associato al contatore; nel caso invece di un terminale di tipo TRAX verrà restituito il numero di cambiamenti di stato intercorsi dall'ultima interrogazione.
Si noti che passando alla primitiva un valore di parametro non valido, la primitiva terminerà senza eseguire alcuna operazione. Inoltre (se il log è attivo almeno a livello *standard*) verrà registrata una segnalazione di *warning* sul file di log

IOReset(cCounter)

Primitiva che permette di resettare il valore del contatore *cCounter* specificato
La funzione restituisce 1 se il comando è andato a buon fine; un valore negativo altrimenti.

IOLine (nLine)

Primitiva che permette di leggere il valore esistente sulla linea *nLine* associata ad uno dei contatori interni al terminale.
Come in precedenza, il numero di linee valide varia a seconda del modello di terminale utilizzato, secondo lo schema in tabella:

MODELLO	*LINEE*
TRAX	1,2
TRAX/2P	1,2
PROX	1,2,3,4

La funzione restituisce lo stato della linea di input specificata

ReleON(nRelay,nTime)

Primitiva che permette di chiudere un relè *nRelay* associato al terminale per un tempo pari a *nTime* ottavi di secondo.
I relè associabili ad un terminale dipendono dal modello dello stesso secondo la seguente tabella

MODELLO	*RELAY*
TRAX2/P	1,2
PROX	1,2,3

nTime può assumere valori in 1..255: in particolare utilizzando il valore 255 si otterrà una chiusura del relè per un tempo indefinito (fino a quando non si effettuerà una nuova ReleOn).
La funzione restituisce 1 se il comando è andato a buon fine; un valore negativo altrimenti

SendFile (cPCFile, cTermFile)

Primitiva per applicazioni di tipo offline: permette di copiare un file dal disco, il cui percorso è specificato in *cPCFile,* nel file system del terminale e precisamente nel file *cTermFile*.
La funzione restituisce un numero con il seguente significato:

>=0	(pari al numero di record copiati) trasmissione a buon fine
-1	errore di trasmissione o file origine non trovato
-2	errore nella specifica del file di destinazione
-3	errore sul tentativo di scrittura sul drive di destinazione
-4	errore sul cambiamento di terminale

La primitiva è utilizzabile sono con terminali di tipo programmabile

ReceiveFile (cPCFile, cTermFile)

Primitiva per scaricare il contenuto di un file *cTermFile* presente sul file system del terminale in

un file *cPCFile* su disco. La funzione restituisce un numero con il seguente significato:

>=0	(pari al numero di record copiati) trasmissione a buon fine
-1	errore di trasmissione
-2	errore nella specifica del file di destinazione
-3	errore sul tentativo di scrittura sul drive di destinazione
-4	errore sul cambiamento di terminale

La primitiva è utilizzabile sono con terminali di tipo programmabile

DeleteFile(cTermFile)

Primitiva per cancellare un file dal file system del terminale. La funzione restituisce un numero con il seguente significato:

0	trasmissione a buon fine
-1	errore di trasmissione
-2	errore nella specifica del file di destinazione
-3	errore sul tentativo di scrittura sul drive di destinazione
-4	errore sul cambiamento di terminale

La primitiva è utilizzabile sono con terminali di tipo programmabile

File(cTermFile)

Primitiva che testa l'esistenza del file *cTermFile* nel file system del terminale. La funzione restituisce un numero con il seguente significato:

1	il file esiste
0	il file non esiste
-1	errore di trasmissione
-2	errore nella specifica del file di destinazione
-3	errore sul tentativo di scrittura sul drive di destinazione
-4	errore sul cambiamento di terminale

La primitiva è utilizzabile sono con terminali di tipo programmabile

Adir (@aArray,cPattern)

Primitiva che estrae dal terminale tutti i file che corrispondono al pattern *cPattern* e li pone nell'array *aArray*. La funzione restituisce un numero con il seguente significato:

>=0	numero di file trovati
-1	errore di trasmissione
-2	errore nella specifica del file di destinazione
-3	errore sul tentativo di scrittura sul drive di destinazione
-4	errore sul cambiamento di terminale

La primitiva è utilizzabile sono con terminali di tipo programmabile

StopApplication()

Primitiva che *deve* essere utilizzata per terminare i driver che implementano il polling e la gestione dei dati di applicazioni di tipo offline: una volta eseguito il driver, la primitiva riporta il driver stesso in uno stato in cui esso è pronto per la nuova esecuzione (che avverrà al prossimo istante di polling, secondo quanto pianificato dall'opportuno timeout)

SVILUPPO DI DRIVER

Come abbiamo avuto modo evidenziare in precedenza, lo sviluppo di un'applicazione verticale prevede due diversi momenti:

- la definizione di un file .prg contenente l'estensione della classe base dei terminali necessaria per l'applicazione corrente
- la scrittura di un batch CodePainter che implementa l'automa a stati finiti alla base dell'applicazione, di fatto il driver vero e proprio

Al fine di chiarire perfettamente la tecnica di costruzione delle applicazioni discuteremo due esempi di driver che vengono forniti a corredo del Communication Framework.

Interrogazione Dati di Magazzino

Il driver si propone di permettere il seguente tipo di interrogazione: dato un articolo si potrà richiederne il prezzo in base ad un listino.

Presentiamo innanzitutto come deve essere estesa la classe dei terminali (inserita dentro l'area manuale del batch *GSCF_BD1.def*)

```
* --- Area Manuale = Functions & Procedures
DEFINE CLASS cTxGSCF_BD1 As cTX
  w_PREZZO=0
  w_ARTICOLO=space(15)
  w_LISTINO=space(3)
  w_LISTDET=.f.
ENDDEFINE
* --- Fine Area Manuale
```

Tali attributi permettono di salvare le informazioni raccolte durante i vari stati della transazione, in modo da poterli utilizzare per effettuare la lettura del dato voluto.
Si noti in particolare come la classe c*txGSCF_BD1* sia ridefinita per ereditarietà a partire dalla classe base dei terminali *cTX*.

Passiamo ora ad analizzare il batch CodePAINTER che implementa l'applicazione.
La prima parte del batch come al solito è riservata alle dichiarazioni di variabile.
In questa sezione vale la pena soffermarsi sul passaggio dell'oggetto *pTX* che rappresenta il terminale che invoca il batch: in Ad Hoc il terminale viene identificato dichiarandolo come parametro del batch.
Le altre variabili locali sono normali variabili di appoggio usate all'interno dell'applicazione.

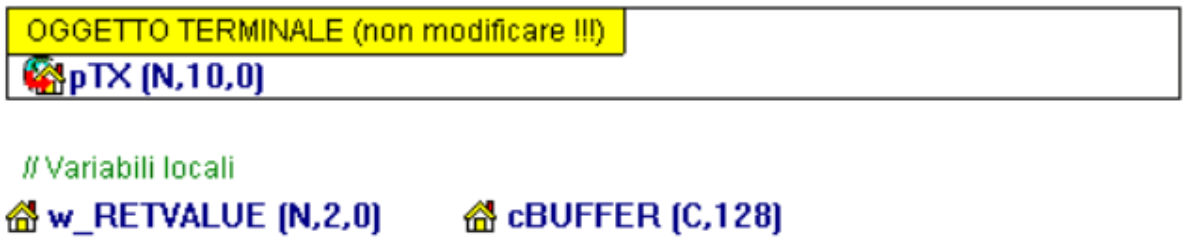


Fig. 3.1 - Dichiarazioni

La seconda parte del batch è riservata alla gestione asincrona dei tasti funzionali: in questa zona vengono gestiti i tasti funzionali che devono essere attivi in qualsiasi momento dell'applicazione. In questo caso ai due tasti considerati vengono associate le seguenti funzionalità: il tasto A riporta semplicemente l'applicazione nello stato 0, mentre il tasto D, oltre a permettere la terminazione della transazione in corso, fissa anche il codice del listino al listino di dettaglio. Al proposito si noti l'uso che viene fatto degli attributi *w_ListDet* e *w_Listino*, definiti nella classe terminale estesa.

Si noti inoltre l'uso della primitiva *FunctionKeyNotManaged* per il trattamento dei tasti funzionali non gestiti: qualora da uno dei terminali si prema un tasto funzionale diverso da A o D il comportamento dell'applicazione sarà quello di emettere un segnale acustico e di ripristinare l'ultimo stato consistente.

GESTIONE TASTI FUNZIONALI
CUSTOM

```
Case pTX.FUNCKEY="FA"
     // Gestione tasto funzionale FA        // Abortisce la transazione in qualsiasi momento
      pTX.w_LISTDET=.F.
      pTX.nSTATO=0

Case pTX.FUNCKEY="FD"
      // Fissa codice listino a dettaglio
      pTX.w_LISTDET=.T.
      pTX.w_LISTINO="DET"
      pTX.nSTATO=0

Case !empty(pTX.FUNCKEY)
      // Tasti funzionali o altri caratteri non gestiti
      pTX.FunctionKeyNotmanaged()
      Stop
End Case
```

Fig. 3.2 – Gestione asincrona tasti funzionali

Descriviamo ora la parte fondamentale del batch, quella che realizza l'automa a stati finiti: questa sezione verrà strutturata come un *case* i cui rami implementano ciascuno uno stato dell'automa.

GESTIONE AUTOMA APPLICAZIONE VERTICALE

```
Case pTX.nStato=0
    w_RETVALUE=pTX.send('Articolo?','XXXXXXXXXXXXXXX')        // Side effect: faccio partire un timeout
    pTX.nSTATO=1
Case pTX.nStato=1
     w_RETVALUE=pTX.receive(@cBUFFER)
     If w_RETVALUE=1
          // Il polling ha dato esito positivo entro il timeout
          pTX.w_ARTICOLO=cBUFFER
          If pTX.w_LISTDET
               // Listino al dettaglio
               page 2:RICERCA PREZZO IN BASE AL LISTINO
               pTX.NextStateAfter(0,3)          // Timeout esplicito di visualizzazione del risultato
               pTX.nSTATO=3
          Else
               // Richiede listino
               w_RETVALUE=pTX.send('Listino?','XXX')
               pTX.nSTATO=2
          End If
     End If
```

Fig. 3.3 – Automa – stati 0,1

Il primo stato, lo stato in cui inizia l'applicazione, dovrà essere necessariamente sempre lo stato 0: in questo caso viene eseguita una *send* che invia al terminale un prompt con la richiesta dell'articolo, seguito da una maschera di input; tale maschera di input viene definita per mezzo di una picture che dovrebbe avere un aspetto familiare per un programmatore Visual FoxPro: infatti con questa sintassi esprimiamo che il codice dell'articolo che dovrà essere inserito potrà essere composto da al più 15 caratteri alfanumerici.
Una volta eseguita la *send* l'applicazione passa nello stato 2. Si noti che in corrispondenza della *send* viene attivato il timeout di verifica definito a livello di anagrafica terminali: il comportamento sarà quello di abortire la transazione e riportare l'applicazione nello stato 0 qualora nessun articolo venga imputato da terminale dopo l'esecuzione della *send* per un tempo superiore al timeout di fallimento.
Tale timeout viene attivato ogni volta che viene eseguita una *send*.

Nello stato 1 l'applicazione effettua una *receive*, il cui esito viene posto nella variabile locale *w_RETVALUE*: finchè non viene ricevuto nulla dal terminale (e finchè non scade il timeout di verifica) l'applicazione rimane nello stato 1, in cui continua ad effettuare il polling del terminale. Quando la *receive* invece ha esito positivo (*w_RETVALUE=1*) allora il dato letto da terminale viene ricevuto nella variabile *c_BUFFER*: si noti come il parametro dentro la *receive* sia passato per riferimento (tramite il prefisso *@*) in modo che il suo valore possa essere modificato all'interno della funzione.
Tale valore viene salvato nella proprietà *w_ARTICOLO,* definita nella classe terminale estesa: questo permetterà di poter riutilizzare in qualunque momento il giusto valore dell'articolo immesso, ogni volta che il batch verrà nuovamente invocato dal terminale attualmente considerato.

```
RICERCA PREZZO IN BASE AL LISTINO
w_MVCODART <= pTX.w_ARTICOLO          // Codice o barcode articolo
w_MVDESART <= ""                      // Descrizione articolo
w_MVCODLIS <= pTX.w_LISTINO           // Codice listino
w_MVQTAMOV <= 0
w_MVDATDOC <= date()
w_PREZZO <= 0
// Determina codice articolo in base al codice a barre
 GSVE_BRA
// Determina prezzo in base al listino
w_PREZZO <= 0
w_vars <= "|w_PREZZO"
w_fields <= "|LIPREZZO"
 GSLU_DPL with "LIS_TINI",w_MVCODART,w_MVCODLIS,w_MVDATDOC
// Informazioni su listino e valuta
 w_VALLIS (N,3,0)      w_VVU (N,3,0)      w_SIMVAL (C,5)
Read from LISTI
Read from VALUT
// Salva prezzo
 pTX.w_PREZZO=w_PREZZO
If w_PREZZO=0
  // Non ci sono corrispondenze di prezzo relativamente ai dati inseriti
   cBUFFER <= "Prezzo non disponibile"
Else
   // Corrispondenza trovata (es. LIT 15.000)
   // Prepara messaggio per display
   cBUFFER <= Left(w_MVDESART+space(pTX.w_TXDISCOL),pTX.w_TXDISCOL)          // Descrizione articolo su prima riga
   cBUFFER <= cBUFFER+alltrim(left(w_SIMVAL,3))+" "+TRANSFORM(w_PREZZO,PS(11,w_VVU))   // Prezzo su riga successiva
End If
 w_RETVALUE=pTX.send(cBUFFER,"_")
```

Fig. 3.4 – GSCF_ED1, pag. 2 – ricerca prezzo in base al listino

In questo caso il significato associato al dato letto dal terminale sarà quello di un codice di ricerca: come si può vedere, se si è scelto di utilizzare il listino di dettaglio (con il tasto funzionale D), verrà effettuata una chiamata alla pag. 2 del batch (riprodotta in fig. 3-4): il codice dell'articolo verrà determinato per mezzo di una chiamata al batch GSVE_BRA e – una volta reperito il codice dell'articolo – in base a questo verranno ricercati il prezzo (nella GSLU_DPL) e la valuta (per mezzo delle due *read* su LISTI e VALUT)

A questo punto, acquisiti i dati, la procedura verificherà se inviare al terminale il prezzo o un messaggio di prezzo non disponibile (qualora l'interrogazione al database abbia restituito un risultato vuoto) ed effettuerà la *send* corrispondente.
In questo caso il messaggio verrà formattato in modo da essere rappresentato in maniera elegante sul display: ricordiamo tuttavia che il Communication Framework effettua sempre una formattazione minima dei messaggi all'interno della primitiva *send*, tenendo conto in modo automatico delle caratteristiche del display del terminale cui è diretto il messaggio.

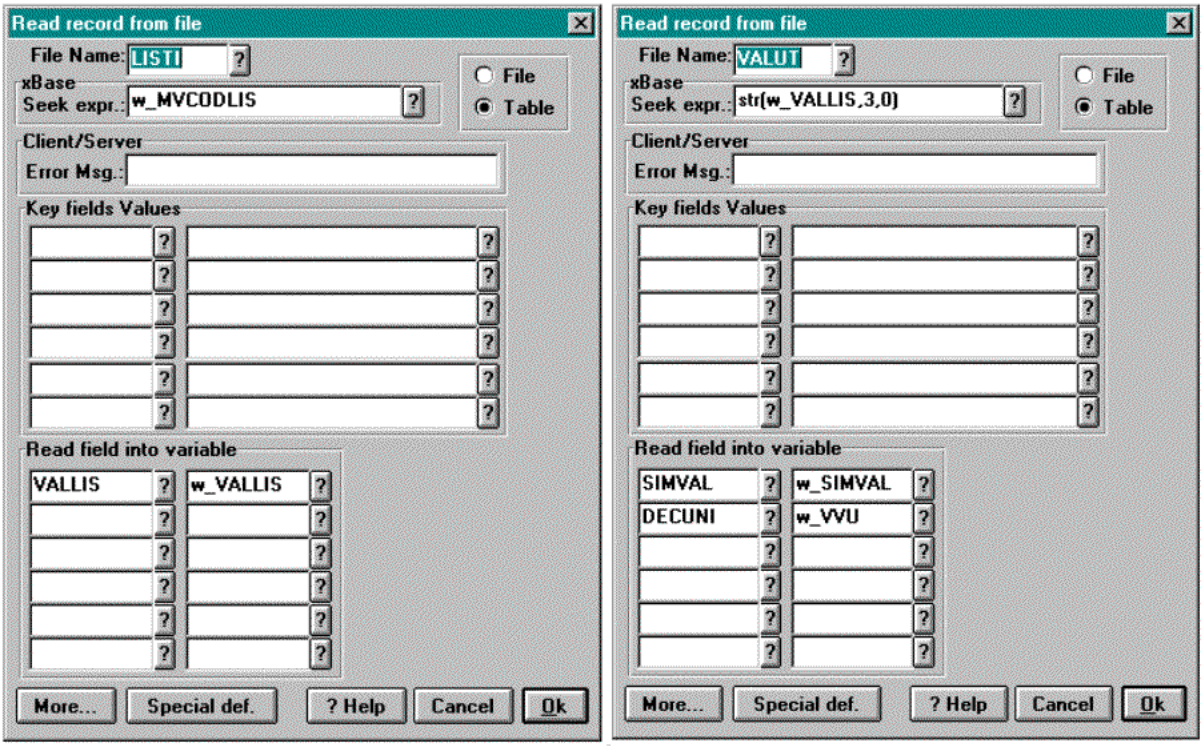


Fig. 3.5 – Automa – Stato 3

```
Case pTX.nStato=3

    // Stato di visualizzazione del risultato: interrotto quando scade il timeout o per pressione del tasto 1 sul terminale
    w_RETVALUE=pTX.receive(@cBUFFER)

    If w_RETVALUE=1
        pTX.nSTATO=0
    End If

End Case
```

Fig. 3.6 – Determinazione della valuta

Si noti come l'applicazione prima di passare nello stato 3, effettui una chiamata alla primitiva *NextStateAfter*: con questa chiamata viene fatto partire un timeout personalizzato (in questo caso pari a 3 secondi), allo scadere del quale l'applicazione passerà automaticamente nello stato specificato nel primo parametro (in questo caso lo stato 0)

Lo stato 3 corrisponde ad uno stato di visualizzazione temporanea del dato: la visualizzazione viene mantenuta fino a quando non venga premuto un qualsiasi tasto sulla tastiera del terminale o non scada il timeout attivato in corrispondenza dell'esecuzione della *NextStateAfter*: per come è stato scritto il codice in questo esempio, in entrambi i casi l'applicazione ricomincerà dallo stato iniziale.
Si noti tuttavia che questo tipo di approccio non è l'unico realizzabile per gestire una visualizzazione temporanea di un dato: ad esempio è possibile far passare l'applicazione in uno stato non definito dell'automa dopo l'esecuzione della *NextStateAfter*, programmando uno dei tasti funzionali, per permettere l'interruzione della visualizzazione prima del timeout impostato con la *NextStateAfter*.

Ritornando allo stato 1, se non si era premuto il tasto funzionale D, l'applicazione richiederà con una nuova *send* il codice del listino da utilizzare.
Questa volta il dato atteso dovrà essere costituito da una stringa composta da al più tre caratteri alfanumerici.
Effettuata la *send*, l'applicazione passerà nello stato 2.

```
Case pTX.nStato=2
    w_RETVALUE=pTX.receive(@cBUFFER)
    If w_RETVALUE=1
        // La receive ha avuto esito positivo entro il timeout
        pTX.w_LISTINO=cBUFFER
        page 2:RICERCA PREZZO IN BASE AL LISTINO

        pTX.NextStateAfter(0,3)          // Timeout esplicito di visualizzazione del risultato
        pTX.nSTATO=3

    End If
```

Fig. 3.7 – Automa – Stato 2

Nello stato 2 l'applicazione si porrà nuovamente in ricezione di un dato dal terminale (il codice del listino): una volta ricevuto (*w_RETVALUE=1*) , il dato verrà salvato nella proprietà *w_LISTINO* della classe terminale estesa dopodichè l'applicazione proseguirà come in precedenza richiamando prima la pagina 2 del batch e poi passando nello stato di visualizzazione (*pTX.nStato=3*)

🗁 Rilevazione Dichiarazioni di Produzione

Analizziamo ora un'applicazione che trova una sua applicazione in contesti di gestione della produzione: vogliamo produrre in automatico le dichiarazioni di produzione a partire da un ODL stampato con gli opportuni codici a barre.\

Iniziamo al solito con il definire l'estensione dello stato dei terminali per ereditarietà dalla classe base *cTX*:

```
* --- Area Manuale = Functions & Procedures
DEFINE CLASS cTxGSCF_BD3 As cTX

  * Informazioni per AVA_PROD
  w_CPCODOPE=space(15)          && Codice operatore
  w_CPDATRIL=date()             && Data Movimento
  w_CPNUMRIL=0                  && Numero Movimento
  w_CPNUMRIG="000"              && Numero Riga
  w_CPCAUAVA=space(3)           && Causale Rilevazione
  w_CPTIPCAU=space(1)           && Tipo Causale
  w_CPINIZIO=0                  && Ora Inizio
  w_CP__FINE=0                  && Ora Fine
  w_CPODPPRI=space(7)           && Codice ODL
  w_CPODPCOM=space(2)           && Competenza ODL
  w_CPSEQUEN=0                  && Fase/Operazione ciclo
  w_CPCODART = space(15)        && Codice articolo
  w_CPUNIMIS = space(2)         && Unita' di misura
  w_CPMAGCAR = space(2)         && Magazzino di carico
  w_CPTIPMOV = space(1)         && Tipo Risorsa
  w_CPCODPRE = space(15)        && Risorsa codice prestazione
  w_CPCODMAC = space(15)        && Risorsa codice macchina
  w_CPCODGRU = space(15)        && Risorsa gruppo macchina
  w_CPQTAMOV = 0                && Qta movimentata
  w_CPQTAEQU = 0                && Qta movimentata 1^UM

  w_CPFLGIAC = space(1)         && Flag Acc/Saldo
  w_CPCAUSCA = space(3)         && Qta movimentata 1^UM
  w_CPQTASCA = 0                && Flag Acc/Saldo
  w_CPFLCARI = space(1)         && Flag Carico Magazzino

ENDDEFINE
* --- Fine Area Manuale
```

Passiamo ora ad analizzare il batch CodePAINTER che implementa il driver (*GSCF_BD3.def*)

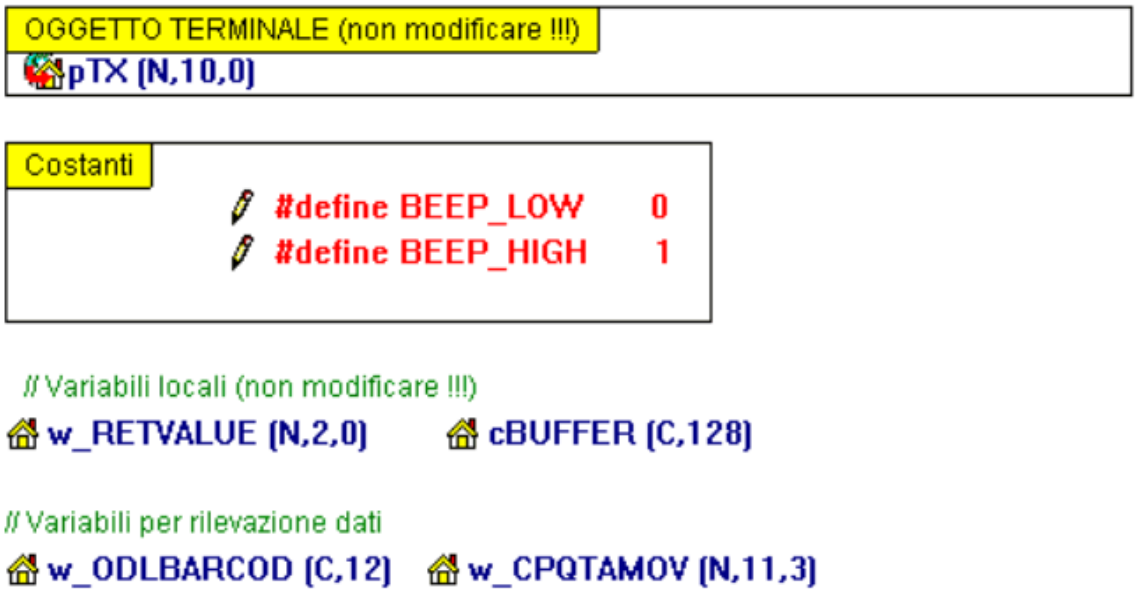


Fig. 3.8 - Dichiarazioni

Per quanto riguarda la parte di dichiarazioni non vi è molto da notare: ancora una volta richiamiamo l'attenzione sul passaggio dell'oggetto terminale chiamante, che rappresenta il dato fondamentale per tutta l'elaborazione.
In questo esempio vengono definite due costanti che serviranno per utilizzare la primitiva *beep.*
Infine, in questo caso, non si definiscono gestioni asincrone per i tasti funzionali.

Passiamo ora ad analizzare l'automa che implementa l'applicazione.

Fig. 3.9 – Automa – stati 0,1

Nello stato iniziale (al solito, sempre lo stato 0) vengono inizializzate le proprietà della classe dei terminali estesa relative al codice operatore (*pTX.w_CPCODOPE*) e al numero di riga (*pTX.w_CPNUMRIG*); viene fatta la *send* con cui si richiede il codice dell'operatore; infine l'applicazione passa nello stato 1.

Nello stato 1, l'applicazione si pone in attesa di un dato dal terminale per mezzo di una *receive*. Quando da terminale viene immesso un dato (*w_RetValue=1*), questo viene salvato nella proprietà della classe dei terminali estesa *pTX.w_CPCODOPE*.
A questo punto viene effettuata una verifica del codice dell'operatore. In questo caso le operazioni di verifica sono state spostate dentro al batch GSPD_BRD.

```
RILEVAZIONE DATI AUTOMATICA

pOPER (C,10)            // Operazioni
pTX (C,10)

pRESULT (C,10)

// Variabili per rilevazione dati
w_ODLBARCOD (C,12)   w_CPODPPRI (C,7)   w_CPODPCOM (C,2)   w_CPSEQUEN (N,3,0)
w_CPCODART (C,15)    w_CPUNIMIS (C,2)   w_CPQTAMOV (N,11,3)
w_CPTIPMOV (C,1)     w_CPCODMAC (C,15)  w_CPCODPRE (C,15)  w_CPCODGRU (C,15)
w_CPTIPCAU (C,1)

Case pOPER="Check_Operatore"
     Read from OPE_RATO
```

Fig. 3.10 – GSPD_BRD - Check Operatore

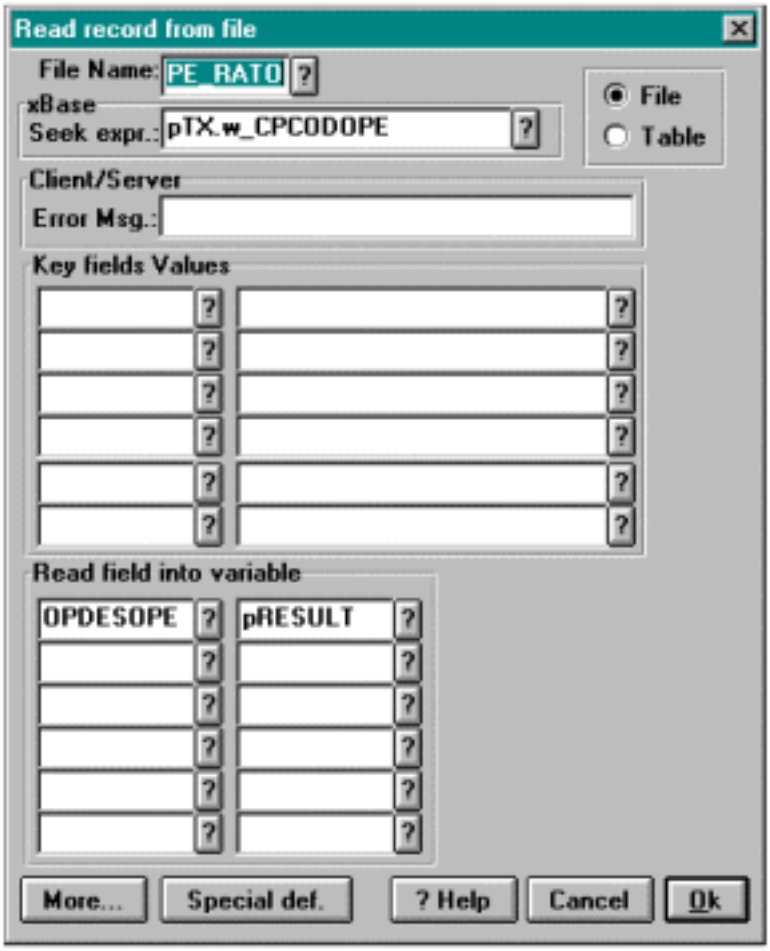


Fig. 3.11 – GSPD_BRD – Check Operatore

Questo tipo di approccio è fortemente consigliato durante la scrittura di driver per il Communication Framework: nel batch GSPD_BRD, infatti, sono state spostate tutte le operazioni di lettura e scrittura nei confronti del database: il vantaggio di questa soluzione si comprende notando che ogni volta che viene invocato un batch generato con CodePAINTER, la procedura apre sempre tutte le tabelle di lavoro, anche quando queste non vengono utilizzate.
Spostando le interazioni con la base di dati in un batch separato, si ottiene il risultato di aprire le tabelle solo quando questo effettivamente serve, ottimizzando le prestazioni: questo aspetto riveste un'importanza fondamentale nel caso del Communication Framework, che ha il compito di gestire reti di dispositivi in tempo reale.

Una volta effettuata la ricerca il controllo ritorna al driver: se l'operatore non è stato riconosciuto la proprietà del terminale viene svuotata, viene inviato al terminale un segnale acustico di errore tramite la primitiva *beep* e l'automa viene riportato nello stato 0 di partenza.
Il segnale acustico è stato impostato su un tono basso per differenziarlo dai normali toni di lavoro del terminale.
Se, al contrario, il codice inserito da terminale è stato riconosciuto come codice valido, l'applicazione provvede a visualizzare sul display del terminale il nome dell'operatore che ha inserito il codice: questo viene fatto inviando il contenuto della proprietà *pTX.w_OPERATORE* con una *send*. Il driver poi si mette in stato di visualizzazione temporanea, attivando un timeout di 1,5 secondi e passando nello stato fittizio 99.
Come già spiegato in precedenza, l'effetto di questa sequenza di operazioni sarà quella di visualizzare per 1,5 secondi il nome dell'operatore sul display; una volta scaduto questo intervallo di tempo, l'applicazione passerà automaticamente nello stato 10 (primo parametro della *NextStateAfter*.

Nello stato 10 l'applicazione invia sul display del terminale un menu a due scelte con cui si invita l'operatore a segnalare se intende iniziare o finire la sessione di lavoro, dopodichè si mette in ascolto di un dato dal dispositivo, passando nello stato 11.

```
Menu opzioni
    Case pTX.nStato=10

        w_RETVALUE=pTX.send('A=Inizio lavoro F=Fine lavoro','')
        pTX.nSTATO=11

    Case pTX.nStato=11
        // Gestione tasti funzionali
        Case pTX.FUNCKEY="FA"                   // Nuovo lavoro
            pTX.nSTATO=20
            // Ore inizio
            w_TIME <= left(time(),5)
            pTX.w_CPINIZIO=val(strtran(w_TIME,":",","))
            w_RETVALUE=pTX.send('INIZIO ORE '+w_TIME,'')
            pTX.NextStateAfter(20,1.5)
            pTX.nSTATO=99               // Stato non esistente, transazione in automatico per timeout 2 sec

            // Reset contatore A
            pTX.IOReset('A')

        Case pTX.FUNCKEY="FF"                   // Fine lavoro
            // Fine lavoro
            w_OK <= .F.
            GSPD_BRD with "Fine_Lavoro",pTX,w_OK
            pTX.nSTATO=0

        Case .T.
            // Tasti funzionali o altri caratteri non gestiti
            pTX.FunctionKeyNotmanaged()
        End Case
```

Fig. 3.12 – Automa – Stati 10,11

Nello stato 11 l'applicazione attende la pressione di un tasto funzionale: ricevendo il tasto funzionale A si inizierà la registrazione dei dati; ricevendo il tasto funzionale F si considererà conclusa la sessione di lavoro (al proposito si ricorda che il tasto funzionale F deve essere digitato due volte per avere effetto: in questo caso è stato utilizzato come controllo di sicurezza per essere sicuri che si voglia concludere davvero la sessione).
La pressione di un qualsiasi altro tasto sul terminale viene intercettata dalla primitiva *FunctionKeyNotManaged*, che invia al terminale una segnalazione acustica di errore e riporta l'applicazione in attesa di una scelta legale.

Qualora sia stata inviata una segnalazione di inizio lavoro, l'applicazione registra l'ora di inizio lavorazione nella proprietà *pTX.w_CPINIZIO*, evidenziandola sul display con una *send* (questa volta il tempo di visualizzazione esplicitato dalla *NextStateAfter* sarà di 1,5 secondi). Dopo aver visualizzato il dato, l'automa passerà nello stato 20. Prima però verrà effettuata una *IOReset* del contatore interno A: l'effetto di questa primitiva è quello di azzerare il contatore A presente sul terminale, in modo da prepararlo a contare automaticamente il numero di pezzi lavorati. Chiaramente per utilizzare questa funzionalità dovrà essere predisposto un dispositivo di contapezzi, provvisto di tutti gli opportuni collegamenti al terminale.

Nello stato 20, l'applicazione invia la richiesta della causale di lavorazione, per cui ci si attende un codice alfanumerico di al più 3 caratteri.
Dopo di che si pone in ascolto del dato passando nello stato 21.

Lo stato 21 è lo stato di ricezione della causale: quando questa viene immessa da terminale, l'applicazione la salve nella proprietà *pTX.w_CPCAUAVA* ed effettua una verifica di correttezza: chiaramente tale verifica richiede di interrogare il database e pertanto, per ottimizzare gli accessi, viene spostata all'interno del batch GSPD_BRD, richiamato con il parametro *Check Causale*.

```
Caricamento riga
    Case pTX.nStato=20

        // Richiesta causale
        w_RETVALUE=pTX.send("Causale?","XXX")
        pTX.nSTATO=21

    Case pTX.nStato=21

        // Ricezione causale
        w_RETVALUE=pTX.receive(@cBUFFER)
        If w_RETVALUE=1
            // Verifica causale
            pTX.w_CPCAUAVA=alltrim(cBUFFER)
            w_OK <= .F.
            GSPD_BRD with "Check_Causale",pTX,w_OK
            If w_OK and pTX.w_CPTIPCAU$"P"          // P=Causale con ODL
                // Causale valida
                pTX.nSTATO=30

            Else                                    // A=Attrezzaggio  // F=Fermo macchina  // altri non gestiti
                // Causale non valida
                pTX.w_CPCAUAVA=space(3)
                pTX.w_CPTIPCAU=space(1)
                w_RETVALUE=pTX.Beep(BEEP_LOW)
                pTX.nSTATO=20
            End If
        End If
```

Fig. 3.13 – Automa – Stati 20,21

Viene pertanto verificato se la causale sia valida e se si tratti di una causale con ODL: in caso affermativo l'applicazione prosegue passando nello stato 30, altrimenti le proprietà codice causale e tipo causale vengono svuotate, viene inviato un segnale acustico che evidenzia l'errore e l'applicazione torna a chiedere la causale ritornando nello stato 20.

```
Case pTX.nStato=30
        // Richiesta odl
        w_RETVALUE=pTX.send("ODL+FASE?",repl("X",12))
        pTX.nSTATO=31
Case pTX.nStato=31
        // Ricezione odl
        w_RETVALUE=pTX.receive(@cBUFFER)
        If w_RETVALUE=1
                // Verifica ODL
                w_ODLBARCOD <= alltrim(cBuffer)
                // ODL
                pTX.w_CPODPPRI=substr(w_ODLBARCOD,3,7)
                pTX.w_CPODPCOM=Left(w_ODLBARCOD,2)
                pTX.w_CPSEQUEN=val(substr(w_ODLBARCOD,11,3))
                w_OK <= .F.
                GSPD_BRD with "Check_Load_ODL",pTX,w_OK
                If w_OK
                        // ODL valido
                        If pTX.w_CPTIPMOV="M"
                                pTX.nSTATO=40
                        Else
                                pTX.nSTATO=50
                        End If
                Else
                        // ODL non valido
                        pTX.w_CPODPPRI=space(7)
                        pTX.w_CPODPCOM=space(2)
                        pTX.w_CPSEQUEN=0

                        w_RETVALUE=pTX.Beep(BEEP_LOW)
                        pTX.nSTATO=30
                End If
        End If
```

Fig. 3.14 – Automa – Stati 30,31

Nello stato 30 viene inviata la richiesta di inserire un codice che identifichi l'ODL e la fase di lavorazione: i codici ammissibili in questo caso saranno stringhe alfanumeriche di lunghezza al più 12.

Nello stato 31, una volta ricevuto il codice, vengono riconosciute le sue componenti: il codice dell'ODL, la competenza e il codice della fase: ancora una volta viene invocato il GSPD_BRD per effettuare gli opportuni test di verifica

```
Case pOPER="Check_Load_ODL"
    w_FOUND1 <= ""
    w_FOUND2 <= ""

    // Legge informazioni ODL e LAVORAZIONI
    Read from ORD_PROD
    Read from LIS_LAVO
    If !empty(w_FOUND1) and !empty(w_FOUND2)
        // OK
        pRESULT <= .T.

        // Quantità da produrre (unità misura ODL)
        w_CPQTAMOV <= w_QTAORD - w_QTAEVA


        // Salva variabili in istanza oggetto terminale
        pTX.w_CPCODART=w_CPCODART
        pTX.w_CPUNIMIS=w_CPUNIMIS
        pTX.w_CPQTAMOV=w_CPQTAMOV

        pTX.w_CPTIPMOV=w_CPTIPMOV
        pTX.w_CPCODPRE=w_CPCODPRE
        pTX.w_CPCODMAC=w_CPCODMAC
        pTX.w_CPCODGRU=w_CPCODGRU

        Case pTX.w_CPTIPCAU="P"          // Causale Produzione con ODL
            pTX.w_CPFLCARI=w_LLFLCARI
        End Case

    End If
```

Fig. 3.15 – GSPD_BRD – CheckLoad ODL ODL

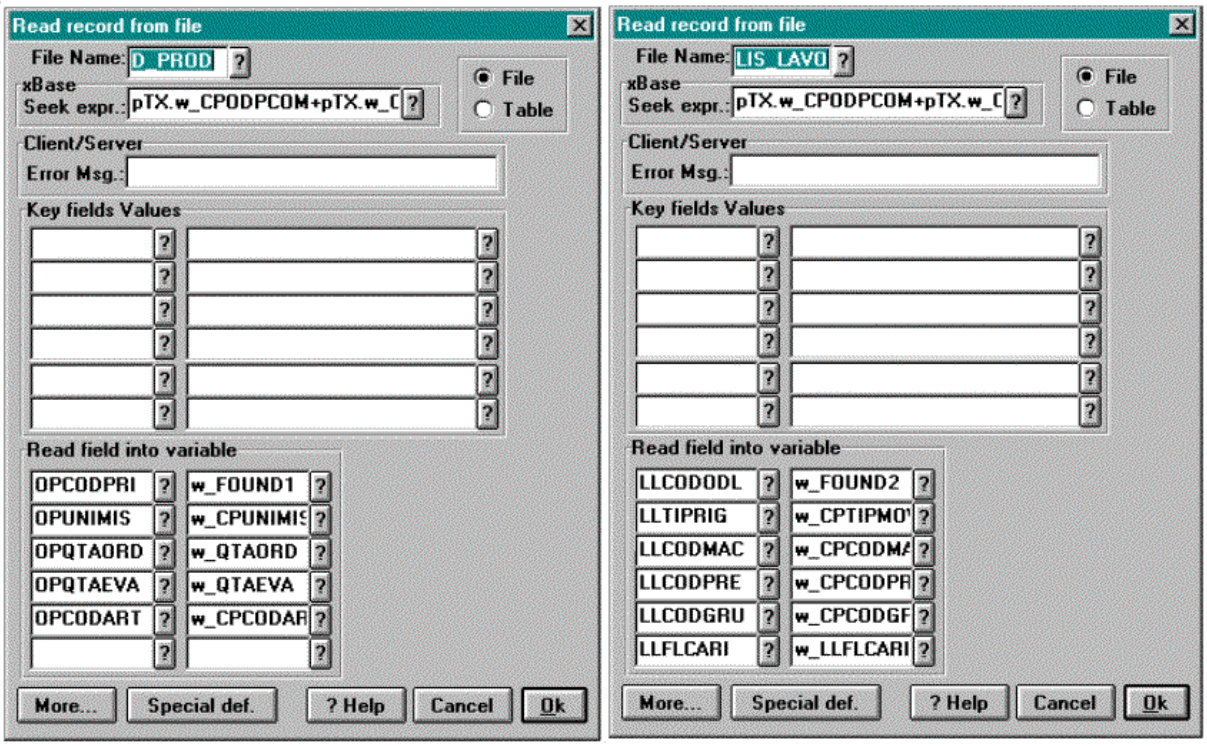


Fig. 3.16 – GSPD_BRD – Check Load ODL

In particolare, se il codice inserito risulta valido, vengono reperite le informazioni riguardanti il codice dell'articolo, l'unità di misura, la quantità da produrre, il tipo di risorsa e i codici di prestazione, macchina e gruppo macchina.
Al rientro nel driver al solito vengono effettuate le operazioni di avanzamento dell'applicazione per un codice valido o di segnalazione di errore e nuova richiesta del codice in caso contrario.

Nello stato 50 viene inviata la richiesta della quantità effettivamente prodotta, che viene attesa nello stato 51: in questo caso si è deciso di lasciare la libertà di registrare il dato in due modi differenti.
L'applicazione è pronta a ricevere il dato per mezzo di un immissione da tastiera grazie alla *receive* nel ramo *.T.* del case.
Tuttavia è possibile immettere il dato anche automaticamente premendo il tasto funzionale C: in questo modo, grazie alla primitiva *IORead*, viene effettuata una lettura del contatore A. Se a tale contatore era stato associato un dispositivo contapezzi, il numero dei pezzi lavorati verrà associato alla variabile *w_CPQTAMOV*.
L'applicazione continua poi confrontando il numero di pezzi lavorati con quello previsto dall'ODL: nel caso in cui sia inferiore, negli stati 60, 61 si richiede se la quantità lavorata debba essere considerata come acconto o se l'ODL debba considerarsi saldato.

```
Case pTX.nStato=50
        // Richiesta quantità
        w_RETVALUE=pTX.send("QTA' PRODOTTA ? "+pTX.w_CPUNIMIS+" ",v_GQ[11])
        pTX.nSTATO=51
Case pTX.nStato=51

        w_CPQTAMOV <= 0
        Case pTX.FUNCKEY="FC"
            // Legge contatore A
            w_CPQTAMOV <= pTX.IORead("A")

        Case !empty(pTX.FUNCKEY)
            // Tasti funzionali o altri caratteri non gestiti
            pTX.FunctionKeyNotmanaged()
            Stop
        Case .T.
            // Ricezione quantità prodotta
            w_RETVALUE=pTX.receive(@cBUFFER)
            If w_RETVALUE=1
                w_CPQTAMOV <= val(cBUFFER)
            End If
        End Case

        If w_CPQTAMOV>0
            If w_CPQTAMOV < pTX.w_CPQTAMOV
                pTX.w_CPFLGIAC="A"
                pTX.nSTATO=60
            Else
                pTX.w_CPFLGIAC="S"
                pTX.nSTATO=70
            End If
            pTX.w_CPQTAMOV=w_CPQTAMOV
        End If
```

Fig. 3.17 – Automa – Stati 50,51

```
Case pTX.nStato=60
        // Richiesta Stato Avanzamento
        w_RETVALUE=pTX.send("1=Saldo,2=Acconto ?","9")
        pTX.nSTATO=61
Case pTX.nStato=61
        // Ricezione Stato Avanzamento
        w_RETVALUE=pTX.receive(@cBUFFER)
        If w_RETVALUE=1
            w_TEST <= val(cBUFFER)
            Case w_TEST=1
                pTX.w_CPFLGIAC="S"
                pTX.nSTATO=70
            Case w_TEST=2
                pTX.w_CPFLGIAC="A"
                pTX.nSTATO=70
            End Case
        End If
Case pTX.nStato=70
        pTX.nSTATO=71
        // Ore fine
        w_TIME <= left(time(),5)
        pTX.w_CP__FINE=val(strtran(w_TIME,":",""))
        w_RETVALUE=pTX.send("FINE ORE "+w_TIME,"")
        // Scrive dati rilevati
        w_OK <= .F.
        GSPD_BRD with "Salva_Dati_Rilevati",pTX.w_OK
        pTX.NextStateAfter(10,1.5)
        pTX.nSTATO=99          // Stato non esistente, transazione in automatico per timeout 2 sec

End Case
```

Fig. 3.18 – Automa – Stati 60,61,70

L'applicazione infine termina rilevando l'ora di fine lavoro (che viene visualizzata sul display) e salvando i dati rilevati.

REGOLE DI SVILUPPO

Al fine di ottenere applicazioni perfettamente funzionanti, lo sviluppo dei driver dovrà seguire alcune semplici regole, che vengono elencate di seguito, atte a garantire la correttezza e la riduzione dei tempi di elaborazione:

- nella sezione dichiarazioni del batch deve esistere la dichiarazione dell'oggetto terminale chiamante: esso deve essere definito come un parametro del batch;
- nella gestione asincrona dei tasti funzionali deve essere previsto un ramo *case .T.* in cui sia contenuta la chiamata alla primitiva *FunctionKeyNotManaged*, che permette il trattamento dei tasti funzionali non gestiti;
- lo stato iniziale dell'applicazione deve essere necessariamente lo stato 0;
- ogni *receive* deve essere seguita da un test del risultato restituito dalla primitiva, al fine di verificare se il polling ha avuto successo o meno;
- per la maggior parte delle applicazioni i driver possono essere strutturati come sequenze strettamente alternate di *send* e *receive*; tuttavia in alcuni casi può essere necessario inviare due *send* consecutive: in tal caso non inserire mai due *send* consecutive nello stesso stato: questo avrebbe l'effetto di sporcare il buffer di trasmissione e quindi bloccare l'applicazione;
- non inserire due *receive* nello stesso stato: poiché in effetti il polling del terminale viene effettuato al di fuori del driver, di fatto la seconda receive non avrebbe mai effetto;
- per ottimizzare i tempi di elaborazione si consiglia di strutturare i driver per mezzo di due batch distinti: un primo batch conterrà lo schema dell'automa che deve implementare l'applicazione, mentre tutte le parti di lettura e scrittura sulla base di dati potrebbero essere spostate in un altro batch. Questa strategia ha il vantaggio di caricare le tabelle necessarie alle letture e scritture solo quando l'applicazione si trova in uno stato in cui questo sia effettivamente necessario.

TABELLA DI RIFERIMENTO TASTI FUNZIONALI

La seguente tabella ha lo scopo di fornire il riferimento per la gestione dei tasti funzionali in fase di scrittura dei driver. Prima di presentare la tabella facciamo alcune precisazioni sui tasti che la compongono:
il tasto funzionale F è particolare perché non invia direttamente un segnale ma prepara alla pressione di un altro tasto (funzionale o no): tali combinazioni sono evidenziate nella tabella dalla coppia F<key>.
Sulla tastiera compaiono due tasti C di cui uno ha il significato di cancellazione: poiché di fatto il firmware non fa differenza nel segnale generato nei due casi, anche i Communication Framework non tiene conto della differenza.

La tabella è composta da due colonne con il seguente significato:
KEY: il tasto o la combinazione di tasti F<key> premuta sul terminale;
CODE: il codice che il Communication Framework utilizza per identificare una combinazione di tasti nella colonna KEY

KEY	CODE
A	FA
B	FB
C	FC
D	FD
FA	FA
FB	FB
FC	FC
FD	FD
F1	F1
F2	F2
F3	F3
F4	F4
F5	F5
F6	F6
F7	F7
F8	F8
F9	F9
↔	FREV
F↔	FREV
FF	FF
↑	FUP
F↑	FUP
↓	FDOWN
F↓	FDOWN
FC	FC
↵	FENTER
F↵	FENTER

CONFIGURAZIONE HARDWARE

Come abbiamo già detto in precedenza, una descrizione delle caratteristiche hardware relative ai dispositivi di rilevazione dati Zucchetti TMC esula dagli scopi del presente manuale (al proposito si invita a fare riferimento alla manualistica tecnica rilasciata da Zucchetti TMC). Tuttavia in questa sezione vengono proposte alcune impostazioni di base che dovrebbero garantire il corretto funzionamento dei terminali.

Impostazioni Porta Seriale

Si consiglia di disattivare il flag “Buffer FIFO” nelle impostazioni relative alla porta seriale cui è collegata la rete di terminali Zucchetti TMC. I test effettuati su diverse configurazioni CPU + sistema operativo, hanno indicato che nella maggior parte dei casi il settaggio di tale parametro non ha particolari conseguenze sul comportamento del Communication Framework; tuttavia lasciando attivo il buffer FIFO sulla porta, in alcuni casi relativi a collegamenti con workstation particolarmente veloci è stata evidenziata qualche piccola anomalia di comunicazione.

Parametri Firmware Terminali Zucchetti tmc

TRAX			
PAR	*SIGNIFICATO*	*VAL*	*NOTE*
com port	protocollo usato	0	impostare il valore 0 per connessioni multipunto tramite reti net 92; impostare il valore 1 per connessioni seriali point-to-point rs232 effettuate per mezzo di un adattatore current loop
net92 baud	baud rate per net92	3	la porta di comunicazione seriale asincrona net92 può essere impostata alle seguenti velocità: 1 → 9600 baud 2 → 19200 baud 3 → 57600 baud

TRAX2P			
PAR	*SIGNIFICATO*	*VAL*	*NOTE*
13	com1in	0	definisce il trattamento dei messaggi ricevuti dalla porta com: il valore 0 permette di passare i messaggi all'interprete dei comandi
18	netmode	0	pone il terminale in stato completamente passivo, permettendo il corretto funzionamento del trax programmabile in modo on line
33	inputmode	0	definisce come trattare gli eventi generati dagli input: i bit 0,1 devono essere sempre posti a 0; se i bit 4,5 sono posti a 1 allora gli eventi della corrispondente linea di input (in1, in2) vengono registrati nel file inputs
36	input1	>0	base di conteggio delle variazioni su in1 per generare un evento (0 = disabilitato)
37	input2	>0	base di conteggio delle variazioni su in2 per generare un evento (0 = disabilitato)
65	inputctrl	1	gestione degli eventi generati sugli input digitali: inpostando il valore a 1, al reset di un contatore minore, il terminale lo ricarica ed incrementa il corrispondente contatore maggiore. se col parametro 33 si è abilitata la registrazione degli eventi, ogni evento viene registrato con l'ora corrispondente
66	waitonlin	255	definisce il massimo intervallo di tempo (in 1/8 di secondo) entro cui il computer host deve interagire con il terminale quando questo opera in modo on line. se tale tempo scade, il terminale ritorna in modalità off line. 255 è il valore massimo impostabile e corrisponde a circa 32 secondi
69	inputedge		determina il funzionamento del decremento dei contatori associati alle linee di input. valido a bit: se il bit di un input vale zero il decremento è effettuato sia sulle transizioni on-off che off-on; se vale uno, il decremento è effettuato solo sulle transizioni off-on. bit 0,1,2,3; rispettivamente per in4,in3,in2,in1


Note