W-LINK: un semplice programma di comunicazione per Windows

La API di Windows comprende una quindicina di funzioni mediante le quali è possibile gestire una comunicazione attraverso le porte seriali; dopo averle passate in rassegna nelle ultime due puntate, vi propongo ora un esempio concreto del loro uso mediante un semplice programma di comunicazione. Il programma è ridotto all'essenziale, ma comprende, accanto all'illustrazione «sul campo» di quelle funzioni, anche altri aspetti che mi auguro interessanti, quali l'uso di un file di inizializzazione e della libreria COMMDLG.DLL

di Sergio Polini

Vi sono diversi programmi di comunicazione per Windows, alcuni dei quali ottimi e dotati di numerose e interessanti potenzialità; risulta difficile accingersi alla realizzazione di qualcosa di simile, sia pure in scala molto ridotta, rispettando un obiettivo di concisione coerente con lo spazio e i tempi della rubrica. In un primo momento, quindi, avevo optato per una soluzione drastica; stavo per proporvi il programma di comunicazione più scarno del mondo, quello mostrato in una figura dell'articolo di maggio: niente menu, niente dialog box, nessuna possibilità di configurazione se non la modifica del codice.

Era un po' troppo. Rendere configurabile il programma, inoltre, poteva anche voler dire soffermarsi un attimo su tecniche utili in ogni programma e, forse, non a tutti familiari. Il programma ha quindi ora un semplice menu, consente di impostare e salvare in un file di inizializzazione i parametri di comunicazione, permette la scelta del carattere. La nostra prima tappa consisterà proprio nell'illustrazione di questi aspetti, mediante una prima versione funzionante del programma, limitata alla impostazione della comunicazione e del suo apparire sul video.

Il file di inizializzazione

Si dice che un'immagine vale più di mille parole; preferisco, tuttavia, proporvi nelle figure 1 e 2 la descrizione del menu e della dialog box del programma W-LINK, così come prodotti dal Resource Workshop scegliendo l'opzione *Edit as text*. In questo modo, vi sarà più semplice riprodurli.

Nella figura 3 trovate invece un tipico file W-LINK.INI. Vi sono diverse possibilità per un file di inizializzazione: una sezione di WIN.INI, un file .INI separato, un file ASCII o binario gestito autonomamente. Per gestire un file .INI si dispone di apposite funzioni della API: *WriteProfileString, GetProfileString e GetProfileInt* sono usate per scrivere in una sezione di WIN.INI e per rileggere poi i valori lì annotati; *WritePrivateProfileString, GetPrivateProfileString e GetPrivateProfileInt* sono usate per scrivere in una sezione di un file .INI separato, il cui nome va indicato nell'ultimo parametro. Un'applicazione dovrebbe sempre usare un proprio file di inizializzazione, sia per motivi di efficienza (si evita di costringere Windows a leggersi tutto WIN.INI), sia per semplificare la vita all'utente (in caso di disinstallazione, è più semplice cancellare un file che modificare WIN.INI). Conviene, inoltre, indicare solo il nome del file, senza path; così facendo, si assume che il file sia nella directory di Windows e, in fase di scrittura, se non esiste viene creato.

Nella figura 4 potete trovare la unit COMMWIN, limitatamente a quelle parti che rendono possibile compilare ed eseguire una prima versione del programma W-LINK (figura 5).

Il codice del file W-LINK.PAS non fa altro che verificare che si stia operando sotto Windows 3.1 (ci occorrerà la funzione *EnableCommNotification*) e aprire una finestra istanza della classe *TCommWindow*. L'interface della unit dichiara alcune costanti corrispondenti ad opzioni del menu e la classe *TCommWindow*, limitatamente al constructor e ai metodi corrispondenti a

```
WLINKMENU MENU
BEGIN
      POPUP "&Collegamenti"
      BEGIN
            MENUITEM "&Apri", 101
            MENUITEM "&Chiudi", 102
            MENUITEM SEPARATOR
            MENUITEM "&Esci", 24340
      END
      POPUP "&Opzioni"
      BEGIN
            MENUITEM "&Parametri comunicazione...", 201
            MENUITEM "&Carattere...", 202
            MENUITEM SEPARATOR
            MENUITEM "&Salva", 203
            MENUITEM "&Ripristina", 204
      END
END
```

Figura 1
Il menu del programma W-LINK.

quelle opzioni; vi è però una sezione private, nella quale vediamo, per ora, tre variabili d'istanza e due metodi. Le variabili vengono usate per il nome del file di inizializzazione e per il tipo di carattere che l'utente vorrà usare, i due metodi leggono e scrivono il file di inizializzazione.

Il constructor di *TCommWindow* attiva il menu e legge il file di inizializzazione dopo averne tratto il nome dal parametro *ATitle*. Il metodo *ReadProfile*, che viene chiamato anche da *CMRestoreSetup*, cerca il file .INI e ne legge i valori (se non lo trova, usa i valori di default indicati nel terzo parametro delle funzioni *GetPrivateProfileString*) per assegnarli ai campi di una variabile *CommTransBuf*, usata anche per l'impostazione interattiva dei parametri di comunicazione mediante una dialog box; il metodo *WriteProfile*, chiamato da *CMSaveSetup*, usa la funzione *WritePrivateProfileString* per scrivere nel file W-LINK.INI i valori dei campi «pari» (quelli il cui nome termina con *Sel*) di *CommTransBuf*.

```
WLINKDLG DIALOG 18, 18, 199, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Parametri di comunicazione"
BEGIN
  CONTROL "&Porta:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,4,9,21,8
  CONTROL "",101,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,54,8,31,48
  CONTROL "&Velocità:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,4,25,29,8
  CONTROL "",102,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_VSCROLL|WS_TABSTOP,52,24,33,65
  CONTROL "P&arità:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,4,41,23,8
  CONTROL "",103,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_VSCROLL|WS_TABSTOP,36,40,49,46
  CONTROL "Bit di &dati:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,
    98,9,34,8
  CONTROL "",104,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,175,8,21,48
  CONTROL "Bit di &stop:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,
    98,25,37,8
  CONTROL "",105,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,171,24,25,43
  CONTROL "&Handshake:",-1,"STATIC",SS_LEFT|WS_CHILD|WS_VISIBLE,
    98,41,40,8
  CONTROL "",106,"COMBOBOX",CBS_DROPDOWNLIST|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,147,40,49,33
  CONTROL "OK",1,"BUTTON",BS_DEFPUSHBUTTON|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,53,67,30,14
  CONTROL "Annulla",2,"BUTTON",BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE
    |WS_TABSTOP,115,67,30,14
END
```

Figura 2 - La dialog box per l'impostazione dei parametri di comunicazione.

Impostazione dei parametri di comunicazione

La sezione implementation della unit COMMWIN inizia con la dichiarazione di alcune costanti; con le prime sei di esse vengono identificati i controlli della dialog box dedicata alla impostazione interattiva dei parametri di comunicazione, con le altre si predispongono le stringhe per la selezione del nome della porta, della velocità di trasmissione, della parità, dei bit di dati e di stop e del tipo di *handshake* mediante quei controlli, tutti *drop down list combo box*.

Per poter leggere le selezioni operate dall'utente, si usa il meccanismo del *transfer buffer*; viene quindi dichiarata una classe *TCommSetupDialog* che, in quanto derivata da *TDialog*, possiede una variabile d'istanza *TransferBuffer*; a questa variabile, di tipo *Pointer*, va assegnato l'indirizzo di un **record** contenente campi corrispondenti ai controlli della dialog box. Nel caso di drop down list combo box, per ogni controllo vanno previsti una stringa e un puntatore ad una collezione di stringhe, l'una per la stringa selezionata dall'utente, l'altra per l'insieme delle selezioni possibili. Il **record** *TCommTransBuf* ha quindi dodici campi; quelli «dispari» sono i puntatori alle collezioni, quelli «pari» sono stringhe.

Per le collezioni sarebbe disponibile la classe *TStrCollection* che tuttavia, essendo derivata da *TSortedCollection*, manterrebbe le stringhe in ordine alfabetico. Ho preferito quindi derivare una classe *TSCollection* direttamente da *TCollection*; poiché si assume, normalmente, chè gli elementi compresi nella collezione siano derivati da *TObject*, per elementi di natura diversa occorre ridefinire il metodo *FreeItem* (anche *GetItem* e *PutItem* nel caso si volessero usare i metodi *Load* e *Store*). È prevista una sola variabile di tipo *TCommTransBuf*, i cui puntatori a collezioni vengono inizializzati nel blocco **begin...end** posto alla fine della **implementation**.

Per utilizzare il meccanismo del transfer buffer, non è sufficiente associare alla dialog box la risorsa contenuta nel file W-LINK.RES; occorre anche associare ad ogni controllo l'istanza di una classe. A ciò provvede il contructor *TCommSetupDialog.Init*, che crea tali istanze mediante chiamate di *New* e del contructor *InitResource*.

```
[Setup]
Port=COM1
BaudRate=9600
Parity=Nessuna
DataBits=8
StopBits=1
Handshake=Hardware
```

Figura 3 - Un esempio del contenuto del file di profilo W-LINK.INI.

La dialog box si apre quando l'utente sceglie l'opzione *Parametri di comunicazione* del menu *Opzioni*, mediante il metodo *CMCommSetup*E. Qui ci si limita a creare l'istanza della classe *TCommSetupDialog* da associare alla risorsa WLINKDLG e ad «eseguirla» mediante *TApplication.ExecDialog*. Se l'utente preme il pulsante OK, i campi della variabile *CommTransBuf*, puntata dalla variabile d'istanza *TransferBuffer* della dialog box, vengono aggiornati con i valori selezionati mediante le combo box di questa.

Il funzionamento del meccanismo può essere agevolmente verificato. Compilato il programma, lo si esegue una prima volta aprendo la dialog box, che proporrà i valori di default specificati in *TCommWindow.ReadProfile: PortName[0]* ('COM1'), *BaudRate[6]* ('9600'), *Parity[0]* ('Nessuna'), *DataBits[3]* ('8'), *StopBits[0]* ('1') e *Handshake[1]* ('Hardware'). Chiudendo la dialog box senza cambiare i valori e scegliendo quindi Salva dal menu *Opzioni*, si provocherà la creazione di un file W-

LINK.INI nella directory di Windows uguale a quello riprodotto nella figura 3. Uscendo dal programma ed eseguendolo poi nuovamente, si potrà provare a cambiare i valori di default (letti questa volta da W-LINK.INI), a salvare e ad uscire ancora. Una successiva esecuzione presenterà nella dialog box i nuovi valori per i parametri di comunicazione, letti dal file W-LINK.INI modificato.

Scelta del carattere

Come vedremo meglio il mese prossimo, il dialogo che l'utente vuole condurre durante la comunicazione viene

```
unit CommWin;
interface
uses WinTypes, WinProcs, Objects, OWindows, ODialogs, Strings, CommDlg;
const
  cm_CommSetup = 201;
  cm_FontSelect = 202;
  cm_SaveSetup = 203;
  cm_RestoreSetup = 204;
type
  PCommWindow = ^TCommWindow;
  TCommWindow = object(TWindow)
    constructor Init(AParent: PWindowsObject; ATitle: PChar);
    procedure CMCommSetup(var Msg: TMessage);
      virtual cm_First + cm_CommSetup;
    procedure CMFontSelect(var Msg: TMessage);
      virtual cm_First + cm_FontSelect;
    procedure CMSaveSetup(var Msg: TMessage);
      virtual cm_First + cm_SaveSetup;
    procedure CMRestoreSetup(var Msg: TMessage);
      virtual cm_First + cm_RestoreSetup;
  private
    Profile: array[0..12] of Char;
    LogFont: TLogFont;
    CF: TChooseFont;
    procedure ReadProfile;
    procedure WriteProfile;
  end;
implementation
const
  id_Port        = 101;
  id_BaudRate    = 102;
  id_Parity      = 103;
  id_DataBits    = 104;
  id_StopBits    = 105;
  id_Handshake   = 106;
  PortName: array[0..3] of array[0..4] of Char =
    ('COM1', 'COM2', 'COM3', 'COM4');
  BaudRate: array[0..9] of array[0..5] of Char =

('110','300','600','1200','2400','4800','9600','14400','19200','38400');
  Parity  : array[0..4] of array[0..7] of Char =
    ('Nessuna', 'Dispari', 'Pari', 'Mark', 'Spazio');
  DataBits: array[0..3] of array[0..1] of Char = ('5', '6', '7', '8');
  StopBits: array[0..2] of array[0..3] of Char = ('1', '1,5', '2');
  Handshake: array[0..1] of array[0..8] of Char =
    ('XON/XOFF', 'Hardware');
type
  PCommSetupDlg = ^TCommSetupDlg;
  TCommSetupDlg = object(TDialog)
    constructor Init(AParent: PWindowsObject; AName: PChar);
  end;
  PSCollection = ^TSCollection;
  TSCollection = object(TCollection)
    procedure FreeItem(Item: Pointer); virtual;
  end;
  TCommTranSBuf = record
    PortStrings: PSCollection;
    PortSel: array[0..4] of Char;
    BaudStrings: PSCollection;
    BaudSel: array[0..5] of Char;
    ParityStrings: PSCollection;
    ParitySel: array[0..7] of Char;
    DataStrings: PSCollection;
    DataSel: array[0..1] of Char;
    StopStrings: PSCollection;
    StopSel: array[0..3] of Char;
    HShakeStrings: PSCollection;
    HShakeSel: array[0..8] of Char;
  end;
var
  CommTranSBuf: TCommTranSBuf;
constructor TCommSetupDlg.Init(AParent: PWindowsObject; AName: PChar);
var
  CB: PComboBox;
begin
  inherited Init(AParent, AName);
  New(CB, InitResource(@Self,id_Port,SizeOf(CommTransBuf.PortSel)));
  New(CB, InitResource(@Self,id_BaudRate,SizeOf(CommTransBuf.BaudSel)));
  New(CB, InitResource(@Self,id_Parity,SizeOf(CommTransBuf.ParitySel)));
  New(CB, InitResource(@Self,id_DataBits,SizeOf(CommTransBuf.DataSel)));
  New(CB, InitResource(@Self,id_StopBits,SizeOf(CommTransBuf.StopSel)));
  New(CB,
    InitResource(@Self,id_Handshake,SizeOf(CommTransBuf.HShakeSel)));
  TransferBuffer := @CommTransBuf;
end;
procedure TSCollection.FreeItem(Item: Pointer);
begin
  StrDispose(PChar(Item));
end;

constructor TCommWindow.Init(AParent: PWindowsObject; ATitle: PChar);
begin
  TWindow.Init(AParent, ATitle);
  Attr.Menu := LoadMenu(HInstance, 'WLINKMENU');
  StrLCopy(Profile, ATitle, 8);
  Profile[7] := #0;
  StrCat(Profile, '.INI');
  ReadProfile;
end;

procedure TCommWindow.ReadProfile;
begin
  with CommTransBuf do begin
    GetPrivateProfileString('Setup', 'Port', PortName[0], PortSel,
      5, Profile);
    GetPrivateProfileString('Setup', 'BaudRate', BaudRate[6], BaudSel,
      6, Profile);
    GetPrivateProfileString('Setup', 'Parity', Parity[0], ParitySel,
      8, Profile);
    GetPrivateProfileString('Setup', 'DataBits', DataBits[3], DataSel,
      2, Profile);
    GetPrivateProfileString('Setup', 'StopBits', StopBits[0], StopSel,
      4, Profile);
    GetPrivateProfileString('Setup', 'Handshake', Handshake[1],
      HShakeSel, 9, Profile);
  end;
end;
procedure TCommWindow.WriteProfile;
begin
  with CommTransBuf do begin
    WritePrivateProfileString('Setup', 'Port', PortSel, Profile);
    WritePrivateProfileString('Setup', 'BaudRate', BaudSel, Profile);
    WritePrivateProfileString('Setup', 'Parity', ParitySel, Profile);
    WritePrivateProfileString('Setup', 'DataBits', DataSel, Profile);
    WritePrivateProfileString('Setup', 'StopBits', StopSel, Profile);
    WritePrivateProfileString('Setup', 'Handshake', HShakeSel, Profile);
  end;
end;
procedure TCommWindow.CMCommSetup(var Msg: TMessage);
var
  D: PCommSetupDlg;
  Result: Integer;
begin
  D := New(PCommSetupDlg, Init(@Self, 'WLINKDLG'));
  if Application^.ExecDialog(D) = idOk then ;
end;
procedure TCommWindow.CMFontSelect(var Msg: TMessage);
begin
  FillChar(CF, SizeOf(CF), 0);
  CF.lStructSize := SizeOf(CF);
  CF.hwndOwner := HWindow;
  CF.lpLogFont := @LogFont;
  CF.Flags := CF_TTONLY or CF_SCREENFONTS or CF_EFFECTS;
  CF.nFontType := SCREEN_FONTTYPE;
  ChooseFont(CF);
end;
procedure TCommWindow.CMSaveSetup(var Msg: TMessage);
begin
  WriteProfile;
end;
procedure TCommWindow.CMRestoreSetup(var Msg: TMessage);
begin
  ReadProfile;
end;

var
  i: Integer;
begin
  with CommTransBuf do begin
    New(PortStrings, Init(4, 0));
    for i := 0 to 3 do
      PortStrings^.Insert(StrNew(PortName[i]));
    New(BaudStrings, Init(10, 0));
    for i := 0 to 9 do
      BaudStrings^.Insert(StrNew(BaudRate[i]));
    New(ParityStrings, Init(5, 0));
    for i := 0 to 4 do
      ParityStrings^.Insert(StrNew(Parity[i]));
    New(DataStrings, Init(4, 0));
    for i := 0 to 3 do
      DataStrings^.Insert(StrNew(DataBits[i]));
    New(StopStrings, Init(3, 0));
    for i := 0 to 2 do
      StopStrings^.Insert(StrNew(StopBits[i]));
    New(HShakeStrings, Init(2, 0));
    for i := 0 to 1 do
      HShakeStrings^.Insert(StrNew(Handshake[i]));
  end;
end.
```

Figura 4 - La unit COMMWIN, limitatamente alle parti che consentono di realizzare una prima versione del programma W-LINK.

visualizzato in una finestra di 25 righe per 80 colonne. Usando un carattere «normale», quale OEM_FIXED_FONT, ciò comporta che la finestra, nel caso di un video VGA, occupi quasi tutta la larghezza dello schermo. Si potrebbe quindi desiderare un carattere diverso, magari *true type*, per contenere le dimensioni della finestra.

Tradizionalmente, per usare più tipi di carattere, si disponeva di due soluzioni; si potevano proporre in una dialog box i nomi dei diversi caratteri disponibili mediante la funzione *EnumFonts*, badando a distinguere tra caratteri raster (disponibili in varie dimensioni predeterminate) o scalabili, *device* (tipicamente quelli propri della stampante) o GDI, *true type* o no; si poteva in alternativa mettere a punto la definizione logica del carattere, avvalorando i campi di un record di tipo *TLogFont*, per «creare» il carattere mediante la funzione *CreateFontIndirect*, lasciando a Windows il compito di scegliere il carattere, tra quelli disponibili, più simile alla definizione data.

Con Windows 3.1, si può tuttavia percorrere un più semplice cammino. Si dispone infatti di COMMDLG.DLL, una libreria contenente quanto necessario per implementare con grande semplicità i tipi più comuni di dialog box, eliminando la necessità di definirne l'apparenza visiva e riducendo al minimo il codice necessario alla loro gestione. Si ha così accesso a dialog box per la scelta dei colori, per aprire e salvare un file, per impostare la stampante e poi stampare, per cercare e sostituire stringhe di testo, ed anche per la scelta dei caratteri.

La classe *TCommWindow* ha una variabile d'istanza privata *LogFont*, destinata a tenere memoria del carattere scelto dall'utente, ed una *CF* di tipo *TChooseFont*. La selezione avviene mediante il metodo *TCommWindow.CMFontSelect*, attivato quando si sceglie l'opzione *Carattere* dal menu *Opzioni*. Il metodo si limita al minimo indispensabile, azzerando tutti i campi di *CF* e avvalorandone poi solo alcuni: *lStructSize* con la dimensione della variabile (è necessario), *hwndOwner* con la finestra cui appartiene la dialog box, *lpLogFont* con l'indirizzo della variabile *LogFont*, *Flags* e *nFontType* per le caratteristiche dei caratteri tra cui si vuole che l'utente scelga (nel nostro caso, caratteri per lo schermo, solo *true type*, con possibilità di «effetti» quali barrato, sottolineato e colore). La dialog box viene aperta chiamando la funzione *ChooseFont* che, se l'utente chiude premendo il pulsante OK, aggiorna i valori dei campi di *CF* e *LogFont* secondo le scelte effettuate. Il metodo dovrebbe terminare con qualcosa che provochi l'aggiustamento di quello che si vede sullo schermo, ma di questo parleremo il mese prossimo, come anche della impostazione di un carattere iniziale.

Un argomento come la scelta dei caratteri, infatti, richiederebbe sicuramente più spazio; ho preferito, tuttavia, racchiudere in un'unica puntata la trattazione di temi necessari, ma un po' estranei al filone principale del discorso che stiamo svolgendo. Quando torneremo al problema della gestione della seriale, avendo sott'occhio la versione completa del programma W-LINK, avremo comunque modo di perfezionare alcuni dei meccanismi ora solo abbozzati. MC

```
program WLink;
(*$R W-LINK.RES*)
uses WinTypes, WinProcs, OWindows, CommWin, Strings;
const
  AppName = 'W-Link';
type
  TWLinkApp = object(TApplication)
    constructor Init(AName: PChar);
    procedure InitMainWindow; virtual;
  end;
constructor TWLinkApp.Init(AName: PChar);
var
  Version: Word;
begin
  Version := LoWord(GetVersion);
  if Swap(Version) < $030A then begin
    MessageBox(0, 'Questa applicazione richiede Windows 3.1.', AppName,
               mb_IconStop);
    Halt;
  end;
  Inherited Init(AName);
end;
procedure TWLinkApp.InitMainWindow;
begin
  MainWindow := New(PCommWindow, Init(nil, AppName));
end;
var
  WLinkApp: TWLinkApp;
begin
  WLinkApp.Init(AppName);
  WLinkApp.Run;
  WLinkApp.Done;
end.
```

Figura 5 - Una prima versione del programma W-LINK, limitata alla impostazione dei parametri di comunicazione e alla gestione di un file di inizializzazione.

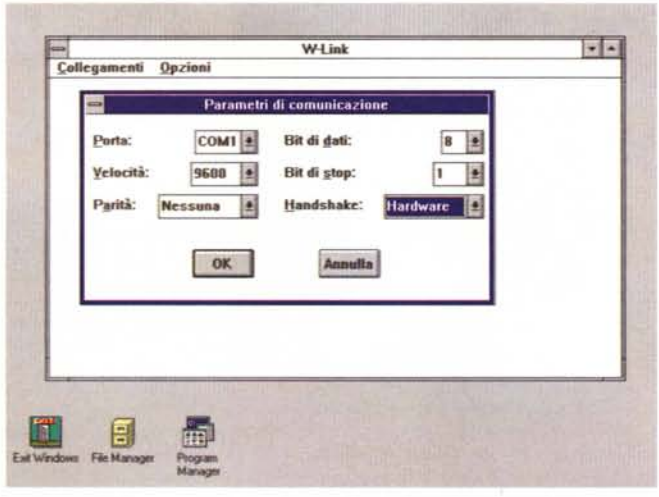

Figura 6 - La dialog box per l'impostazione dei parametri di comunicazione.

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.