Ingenieurbüro für Echtzeitprogrammierung

Mail-LIB

Mail senden unter RTOS-UH

Dok-Rev. 1.2 vom 26.01.2011

Software-Rev. 1.2 vom 25.01.2010

Inhaltsverzeichnis

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	20.07.2006	Kr	Erstellung
1.1	17.08.2006	Ko	Überarbeitet
1.2	25.01.2011	Kr	Klartext bei USER,PASSW bei Angabe einer Mail-Adresse

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

2 Die Mail-Programmierschnittstelle unter RTOS-UH

Der Mail Programmierschnittstelle dient zum Versenden vom E-Mail im Netzwerk oder Internet.

2.1 Die Voraussetzungen der Mail-Programmierschnittstelle

Für die volle Funktionalität müssen im System folgende Voraussetzungen geschaffen sein:

1. Es muss eine `HOSTS`-Datei lokal auf dem Rechner geben.
2. Es muss eine globale Environmentvariable `HOSTFILE` geben, die den Zugriffspfad auf die `HOSTS`-Datei enthält. Die `HOSTS`-Datei muss mindestens den Eintrag für den lokalen Rechner beinhalten.
3. Es sollte eine globale Environmentvariable `SMTP_PROVIDER` geben, die auf eine Provider-Datei zeigt, alternativ kann von Programm die entsprechende Struktur ausgefüllt werden.
4. In der Netzwerkkonfiguration muss eine Route für das Internet bzw. das Netz mit einem SMTP-Server eingerichtet sein, falls er nicht im lokal erreichbaren Netzes liegt.
5. Die Environmentvariable `DNS_SERVER` muss gesetzt sein (siehe stNet2Lib.PDF).

Als Programmiersprache werden PEARL und CREST-C unterstützt.

2.1.1 Provider-Datei

Die lokale Provider-Datei (*/dev/*PROVIDER) hat folgenden Aufbau:

```
base64_user , base64_passw  ; base64 codierte Namen
smtp.xxxx.xxx               ; Name_des_SMTP-server
me@xxxxx.xx                 ; eigene eMail Adresse
Bezeichnung                 ; optional Bezeichnung
<EOF>
```

Der angegebenen Username und das Passwort sind base64 codiert in der Datei abgelegt, sie müssen die Werte enthalten, die für den SMTP-Server beantragt wurden.

! Ab V1.2 kann alternativ der User als eMail-Adresse angegeben werden im Klartext und

dann ist auch das Passwort im Klartext anzugeben.

Die zweite Zeile gibt den Namen des SMTP-Servers an, an den die E-Mail gesendet wird.

In der dritten Zeile wird die Absender und gleichzeitig auch die Antwortadresse angegeben, sie muss auf dem SMTP-Server verfügbar sein.

In der vierten Zeile kann eine optionale Rechnerbezeichnung angegeben werden, die der Empfänger der E-Mail dann in der "VON-Zeile" angezeigt bekommt.

Kommentare sind durch ein Semikolon (;) gekennzeichnet, sie gelten immer für den Rest der Zeile.

! Falls keine Providerdatei erwünscht ist, muss die Provider-Struktur durch das Programm gefüllt werden und nach dem Aufruf der Funktion RT_init_mail_block, der Pointer auf die Provider-Struktur in der mail.provider eingetragen werden.

2.1.2 Environmentvariable SMTP_PROVIDER

Die Environmentvariable SMTP_PROVIDER kann folgendermaßen gesetzt werden:

```
ENVSET -G SMTP_PROVIDER=/R0/PROVIDER
```

Sie gibt an, wo auf dem System die Provider-Datei abgelegt ist, der Pfad */R0/* ist hier beispielhaft angegeben, es kann jeder andere Pfad genutzt werden(z.B. */h0/ETC* oder andere).

2.2 Die MAILLIB Libraries

Für die Sprache PEARL steht die Library MAIL2SRx.P90 zur Verfügung, diese kann auch permanent im Eprom oder Flash abgelegt werden.

Für CREST-C sind die Programme mit der MAILLIB.LIB zu linken.

Alle REF CHAR(1) Typen in PEARL90 müssen mit einem TOCHAR(0) abgeschlossen werden, da die unterlagerten C-Funktionen dies als Stringende-Kennzeichen benötigen.

2.3 Datentypen der Library

Für die Library Funktionen sind einige Datentypen erforderlich, deren Struktur im folgenden aufgeführt wird.

2.3.1 Mail-Block Struktur

Die Definition in PEARL90 sieht folgend aus:

```
TYPE rt_send_mail_block
     STRUCT (/  type       FIXED(15)   , /* Coding */
                stage      FIXED(15)   , /* internal used  */
                timeout    FIXED(31)   ,
                provider   REF rt_provider_data ,
                local_host REF CHAR(1) ,
                sendto     REF CHAR(1) ,
                cc         REF CHAR(1) ,
                betreff    REF CHAR(1) ,
                message    REF rt_mail_body ,
                anhang     REF STRUCT[]
             /) ;
TYPE mail rt_send_mail_block ; /* Kurzform */
```

2.3.2 Mail-Body Struktur

Die Definition in PEARL90 sieht folgend aus:

```
TYPE rt_mail_body STRUCT (/ len FIXED(31), text REF CHAR(1) /) ;
```

2.3.3 Mail-Provider Struktur

Die Definition in PEARL90 sieht folgendermaßen aus:

```
DCL MAIL_MAX_CHAR INV FIXED(15) INIT(80) ;

TYPE rt_provider_data STRUCT (/ user     CHAR( MAIL_MAX_CHAR ) ,
                                passw    CHAR( MAIL_MAX_CHAR ) ,
                                smtp_srv CHAR( MAIL_MAX_CHAR ) ,
                                email    CHAR( MAIL_MAX_CHAR ) ,
                                add_name CHAR( MAIL_MAX_CHAR )
                              /) ;
```

2.3.4 Mail-Anhang Struktur

Die Definition in PEARL90 sieht folgendermaßen aus:

```
TYPE rt_mail_anhang
     STRUCT (/ type    FIXED(15)   , /* Coding */
               reserve FIXED(31)   , /* internal used */
               fname   REF CHAR(1) ,
               path    REF CHAR(1) ,
               next    REF STRUCT[]
             /) ;
```

2.4 Fehlercodes der Library

Der Fehlercode wird in PEARL90 als `FIXED(15)` zurück gegeben.

Es sind folgende Fehlercodes in der Library als Rückgabewert vergeben:

Fehler	Bedeutung	Mail versandt	Funktion
0	OK-Status	Ja	RT_send_mail
-1	LocalHost nicht gesetzt	Nein	RT_send_mail
-2	Provider nicht gesetzt	Nein	RT_send_mail
-3	Keine TCP-Verbindung	Nein	RT_send_mail
-4	SMTP-Server nicht gefunden	Nein	RT_send_mail
-5	TCP Verbindungsabbruch	Eventuell in Teilen	RT_send_mail
-6	Falsche Authentifikation des Users, Passwort nicht akzeptiert	Nein	RT_send_mail
-7	Falscher Absender, wurde nicht akzeptiert	Nein	RT_send_mail
-8	Falscher Empfänger, wurde nicht akzeptiert	Nein	RT_send_mail
-9	Falscher CC, wurde nicht akzeptiert	Nur an Empfänger	RT_send_mail
-11	Interner Fehler, bitte benachrichtigen Sie uns	Nein	RT_send_mail
-12	Anhang nicht gefunden	Ja, ohne Anhang	RT_send_mail
-13	Kein freier Speicher gefunden	-	RT_load_mail_body
-14	Datei mit Text ist leer	-	RT_load_mail_body
-15	Datei nicht gefunden	-	RT_load_mail_body
-16	Lese Fehler in Datei	-	RT_load_mail_body

2.5 Text Codierungen der Mail

Es stehen zur Codierung des Mail-Bodys und des Mail-Anhanges verschiedene Codierungen zur Verfügung.

Die Codierung wird in Mail-Struktur im `TYPE`-Feld angegeben.

Code	Beschreibung	Anwendungen
0	Text (normaler 8 Bit ISO Text)	Mail-Body
1	Text/Html	Mail-Body
2	OCTET/STREAM	Mail-Anhang
3	OCTET/QUOTED	Mail-Anhang
4	ZIP-COMPRESSED (MIME codierter Text)	Mail-Anhang

3 Funktionen der Library im Überblick

Für das Senden einer E-MAIL stehen die folgenden Funktionen in der Library zur Verfügung:

Funktion	Bedeutung
`RT_send_mail`	Senden der E-Mail
`RT_init_mail_block`	Initialisieren des Mail-Blockes
`RT_load_mail_body`	Laden des Mail-Textes aus einer Datei
`RT_free_mail_body`	Freigeben des Mail-Textes
`RT_encode64`	Text nach Base64 wandeln
`RT_decode64`	Base64 nach Text wandeln

Der Präfix *RT_* gilt für die PEARL90 Funktionen, für die CREST-C Funktionen muss er durch den Präfix *rt_* ersetzt werden.

Beispiel:

Aus der Funktion `RT_send_mail` in PEARL90 wird `rt_send_mail` in CREST-C.

3.1 RT_init_mail_block

Voraussetzungen: keine

Die Funktion `RT_init_mail_block` löscht alle Pointer der Mail-Struktur und trägt normalen 8 Bit ISO-Text als Vorbesetzung ein.

3.1.1 Anwendung in PEARL90

```
SPC RT_init_mail_block ENTRY( pmail REF mail )) GLOBAL ;
...
DCL mymail mail    ;
DCL pmail REF mail ;
...
CALL RT_init_mail_block( pmail ) ;
...
```

3.1.2 Anwendung in CREST-C

```
void rt_init_mail_block( mail * pmail ) ;
...
mail  mymail ;
mail *pmail  = &mymail ;
...
len = rt_init_mail_block( pmail ) ;
...
```

3.2 RT_send_mail

Mit dem Aufruf der Funktion `RT_send_mail` wird ein E-Mail verschickt. Dazu sind die folgenden Daten in der Mailstruktur zu setzen:

- `betreff` optional die Mailüberschriftszeile
- `sendto` der Mailempfänger
- `message` optional der Inhalt der Mail
- `anhang` optional, falls eine Datei an die Mail angehängt werden soll

3.2.1 Anwendung in PEARL90

```
SPC RT_send_mail ENTRY( pmail REF mail )
                 RETURNS( FIXED(15) ) GLOBAL ;
...
DCL pmail REF mail ;
DCL errcode FIXED(15) ;
...
errcode = RT_send_mail( pmail ) ;
IF errcode GE 0 THEN
   /* alles OK Mail ist versandt */
ELSE
    /* Fehler aufgetreten */
FIN ;
```

3.2.2 Anwendung in CREST-C

```
short rt_send_mail( mail *pmail ) ;
...
mail * pmail ;
short errcode ;
...
errcode = rt_send_mail( pmail ) ;
if ( errcode >= 0 )
{
   /* alles OK Mail ist versandt */
}
else
{
    / Fehler aufgetreten */
}
```

3.3 RT_load_mail_body, RT_free_mail_body

Die Funktion `RT_load_mail_body` liest aus einer angegebenen Datei den Mailtext ein. Da diese Funktion vom System Speicher zur Zwischenablage des Mailtextes anfordert, muss nach dem Versenden der Mail dieser Speicher mit `RT_free_mail_body` wieder freigegeben werden. Ob der Speicher angefordert worden ist, kann durch den Test auf das Strukturelement `mail.message` getestet werden. Damit ist es möglich, eine Mail mehrfach zu verschicken, ohne den Mailtext mehrfach einlesen zu müssen.

3.3.1 Anwendung in PEARL90

```
SPC RT_load_mail_body ENTRY( pmail REF mail,
                             name CHAR(1) IDENT
                           ) RETURNS( FIXED(15) ) GLOBAL ;
SPC RT_free_mail_body ENTRY( pmail REF mail ) GLOBAL ;
...
DCL pmail REF mail ;
DCL errcode FIXED(15) ;
...
fname = '/R0/meintext' ;
errcode = RT_load_mail_body( pmail, fname.CHAR(1) ) ;
IF errcode GE 0 THEN
   /* alles OK Mailtext eingelesen */
   ...
   /* senden der Mail */
CALL RT_free_mail_body ; /* freigeben des Speichers */
FIN ;
```

3.3.2 Anwendung in CREST-C

```
short rt_load_mail_body( mail *pmail, char *fname ) ;
void  rt_free_mail_body( mail *pmail ) ;
...
mail  *pmail   ;
short  errcode ;
...
errcode = rt_load_mail_body( pmail, "/R0/TEXT") ;
if ( errcode >= 0 )
{
   /* alles OK Mailtext eingelesen */
   ...
   /* Mail versenden */
   rt_free_mail_body( pmail );
}
```

3.4 RT_encode64

Die Funktion RT_encode64 wandelt eine Textstring in einen BASE64-Codierten String um. Es wird der in String bis zur maxlen in den out String umgewandelt. Mit dieser Funktion können die Base64-Codierten Passworte und Usernamen für die Provider-Datei erzeugt werden.

Der out String hat eine um den Faktor 4/3 größere Länge als der in String, der entsprechende Platz muß vorhanden sein.

3.4.1 Anwendung in PEARL90

```
SPC RT_encode64 ENTRY ( ilen   FIXED(15), in CHAR(1) IDENT,
                        out CHAR(1) IDENT, maxlen FIXED(15)
                      ) RETURNS( FIXED(15) ) GLOBAL ;
...
DCL in_len      FIXED(15) ;
DCL out_len     FIXED(15) ;
DCL maxlen      FIXED(15) INIT(STR_CHAR);

DCL in          CHAR(STR_CHAR ) ;
DCL out         CHAR(STR_CHAR ) ;

in = 'meinPasswort' ;
out_len = RT_encode64( LEN(in), in.CHAR(1),
                       out.CHAR(1), maxlen );
PUT 'Base64:<', out, '>' TO TY BY A, A(LEN(out)), A, SKIP ;
```

3.4.2 Anwendung in CREST-C

```
short rt_encode64( short len, char *in,char *out) ;
#define MAX_LEN 80
...
char out [ MAX_LEN ] ;
char *in ;
...
in = “meinPasswort“ ;
out_len = rt_encode64( strlen(in), in, out);
printf(“Base64:<%s>\n“, out ) ;
```

3.5 RT_decode64

Die Funktion RT_decode64 wandelt einen BASE64-Codierten String in einen Textstring um. Es wird der in String bis zur maxlen in den out String umgewandelt.

Der out String hat eine um den Faktor 3/4 kleinere Länge als der in String. Die Aufrufparameter und die Anwendung entspricht der in Kap. 3.4 beschriebenen Funktion RT_encode64.

4 Programmbeispiele

4.1 Senden einer Mail ohne Anhang PEARL90

```
...
TEST_MAIL: TASK ;
  DCL my_mail     mail_block ;
  DCL pMail   REF mail_block ;
  DCL my_text     mail_body  ;
  DCL pText   REF mail_body  ;
  DCL stat        FIXED(15)  ;
  DCL sendto_str  CHAR(80)   ;
  DCL betreff_str CHAR(80)   ;
  DCL text_str    CHAR(80)   ;

  pMail = my_mail ;
  CALL RT_init_mail_block( pMail ) ;
  text_str    = 'dies ist eine TestMail'><TOCHAR(13)><'Hallo'><TOCHAR(13) ;
  sendto_str  = 'walter@test.de'><TOCHAR(0) ;
  betreff_str = 'Eine Mail zum Test'><TOCHAR(0) ;
  pText       = my_text   ;
  pText.text  = text_str.CHAR(1) ;
  pText.len   = LEN( text_str ) ;

  pMail.sendto  = sendto_str.CHAR(1);  /* Empfänger angeben             */
  pMail.betreff = betreff_str.CHAR(1); /* besetzen des Betreff-Feldes   */
  pMail.message = pText ;     /* Einsetzen des Textes als Message-Block */
  pMail.timeout = 20000(31) ; /* maximal 20 Sekunden Timeout           */

  stat = RT_send_mail( pMail ) ; /* Mail verschicken */
  PUT 'Mail:', stat TO TY BY A, F(8), SKIP ;

END ;
```

4.2 Senden einer Mail mit Anhang PEARL90

```
...
TEST_MAIL: TASK ;
 ...
  DCL my_anhang   mail_anhang ;
  DCL pAnhang REF mail_anhang ;
  DCL name        CHAR(64) ;
  DCL path        CHAR(64) ;
...
  wie in 4.1 ...
...

  /* optional einen Anhang dranhängen */
  path = '/H0/DATEN/'><TOCHAR(0) ;
  name = 'MAIL'><TOCHAR(0) ;
  pAnhang.type  = 3 ; /* MAIL_TYPE_OCTET_QUOTED */
  pAnhang.path  = path.CHAR(1) ;
  pAnhang.fname = name.CHAR(1) ;
  pAnhang.next  = NIL ;
  pMail.anhang  = pAnhang ;

  stat = RT_send_mail( pMail ) ;
  PUT 'Mail:', stat TO TY BY A, F(8), SKIP ;

END ;
```