400.50 BIBLIOTHEKEN ALLGEMEIN

400.50.10 Aufbau der OBRZ Runtime Bibliothek

1 Einleitung

Eine runtime bibliothek enthält funktionen, die erst zur laufzeit (runtime) eines programmes (einer applikation) mit diesem zusammengeführt werden (linked), damit ein funktionsfähiges ganzes entsteht. Diese funktionen sind in bibliotheken zusammengefasst (zb plotter-bibliothek).

Die OBRZ runtime bibliothek(en) sind in erster linie für die Fortran umgebung entwickelt. Viele routinen und funktionen können aber auch in anderen umgebungen (Pascal, Assembler, ...) eingesetzt werden.

Wir unterscheiden verschiedene kategorien von subroutinen und funtionen in diesen runtime bibliotheken

- Zu jedem compiler gehört eine runtime bibliothek, welche zb für I/O aufgerufen wird, aber auch die mathematischen basisroutinen enthält. Diese bibliothek enthält also die unterste ebene von rotuinen und funktionen.
- Der kern dieser routinen dient der unterstützung von fehlerbehandlung, trace und debug. Dieser kern wird im vorliegenden kapitel beschrieben.
- Ein precompilers kann wesentlich eleganter arbeiten, wenn er sich auf verschiedne funktionen und subroutinen zur laufzeit eines programmes abstützen kann.
- Der benützer bzw programmierer von applikationen erwartet von einem rechenzentrum verschiedene bibliotheken zur lösung unterschiedlichster aufgaben wie zeichnen, matrix manipulationen, statistische funktionen etc. Im OBRZ bieten wir hierfür:
  - OBRZ PLOT bibliothek, die ein geräte unabhängiges zeichnen erlauben.
  - IMSL (International Mathematical and Statistical Libary) mit derzeit etwa 600 routinen. Diese bibliothek steht auch auf den VAX anlagen des OBK zur verfügung.
  - OBRZ spezials. Hier sind verschiedne dinge zusammengefasst, die grösstenteils zur unterstützung älterer programme notwendig sind, aber auch der vereinfachung von input dienen. Diese subroutinen sind in der allgemeinen OBRZ runtime bibliothek enthalten.

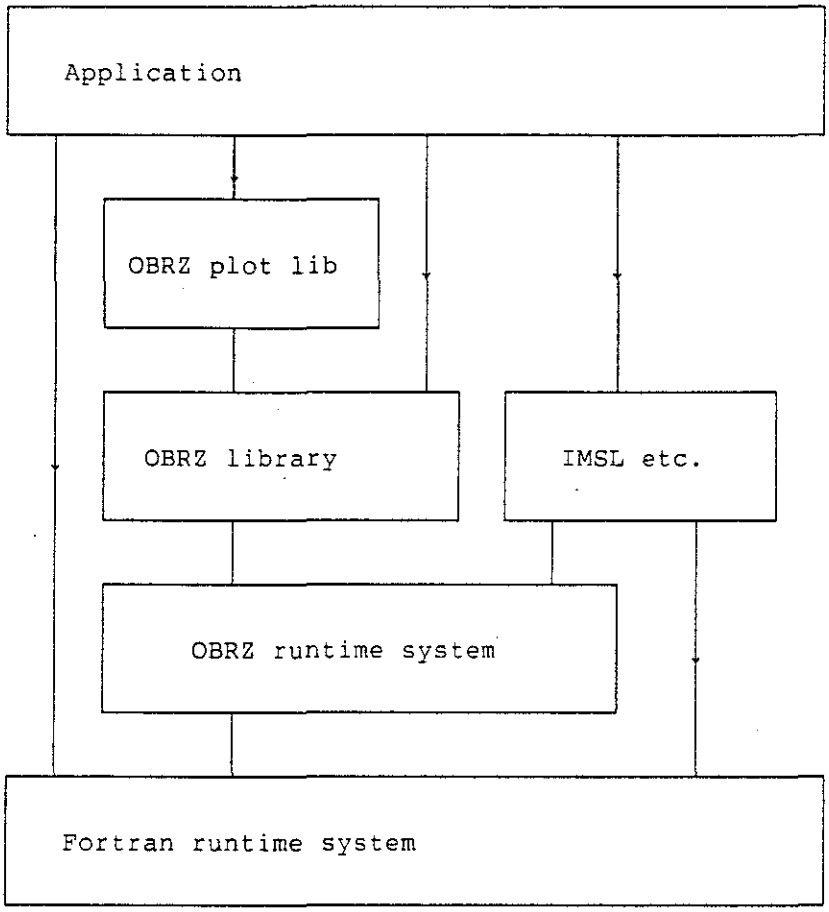
Subroutinen und funktionen für allgemeinen gebrauch müssen sehr zuverlässig arbeiten. Dazu gehört auch, die fehlersuche in programmen zu unterstützen. Die dazu notwendigen funktionen müssen gezielt eingesetzt werden können, müssen also ein- und abschaltbar sein. Auch sollten diese besonderen wirkungen auf einzelne routinen eingeschränkt werden können, damit die übersicht in den meldungen gewahrt bleibt.

Mit einführung des VS Fortran (77) compilers werden die routinen der OBRZ runtime bibliotheken mit folgenden eigenschaften und mechanismen ausgestattet:

- Parameterlisten können beim aufruf der routinen verkürzt werden, indem von hinten her parameter weg gelassen werden können. Dies ist jedoch nur sinnvoll, wenn die entsprechenden defaults genügen. Im allgemeinen werden weggelassene parameter als **undefiniert** behandelt. Je nach funktion kann dies zu einem fehler führen.
- **Fehlende daten** werden als undefiniert (dies ist **nicht** identisch mit null!) behandelt. Die routinen sollen so ausgelegt werden, dass undefinierte daten nicht in die berechnung einbezogen werden. In der behandlung einer tabelle (vektor, matrix) ergeben sich dadurch natürlich auch wieder undefinierte ergebnisse.
- Auf wunsch können bestimmte oder alle routinen **traced** werden. Sie melden sich dann bei ihrem aufruf im allgemeinen mit den aktuellen parametern.
- Auf wunsch geben ausgewählte oder alle routinen während ihrer arbeit **debugging** informationen, insbesondere endergebnisse aus.

- Die routinen überprüfen eingabewerte (parameter, daten), sowie zwischenergebnisse. Fehlermeldungen werden von einer standardroutine MESSAG von einem file gelesen und mit aktuellen werten ergänzt.
- Je nach gewicht des fehlers werden korrigierende aktionen ausgeführt oder die applikation abgebrochen.
- Die ausführlichkeit der meldungen kann bis zur notierung von help-texten gehen. Darin wird die funktion der routine und die art der parameter erklärt (eine kurzgefasste beschreibung).
- Meldungs- und "help"-texte können applikationsbezogen sein und stehen in deutsch oder englisch zur verfügung.

Der gesamte aufbau des OBRZ runtime systems kann folgendermassen dargestellt werden:



1.1 Funktionsbereiche

Diese beschreibung unterscheidet zwischen funktionen für applikationen und funktionen für die runtime bibliothek selbst:

1 Die funktionen für die applikationen dienen dem benützer, welcher routinen und funktionen aus den OBRZ runtime bibliothek im rahmen seiner programme einsetzt. Dies können applikationen sowie unterprogramme für eher allgemeinen zweck sein.

2 In den funktionen für die runtime bibliotheken werden jene sachverhalte beschrieben, die für den aufbau der runtime bibliotheken selbst notwendig sind. Sie dienen also dem entwickler der runtime bibliotheken. Dies gilt insbesondere für die abschnitte "implementierung" und "programmierhilfen".

Im konzept zu den OBRZ runtime bibliotheken wurde wert auf flexibilität gelegt. Daher sind die texte von meldungen von den routinen getrennt. Dies erlaubt, dass die texte je nach umgebung (applikation) anders gestaltet werden können und damit auf die konkrete (fehler)situation besser eintreten können.

2 Deklarationen

In der beschreibung werden folgende deklarationen benützt:

`STRING` ist eine zeichenkette, die in fortran-77 als CHARACTER geeigneter länge zu deklarieren ist. Auch CHARACTER- konstanten ('text') sind damit gemeint.

`INTEGER` ist ein binärer wert, der 1 wort (4 byte) belegt. In fortran wird dies INTEGER*4 genannt.

`LOGICAL` ist ein boolean wert, der 1 wort (4 byte) belegt. In fortran wird dies LOGICAL*4 genannt.

`REAL` ist die darstellung von "gleitkomma zahlen" in einem wort (4 byte); in fortran wird dies als REAL*4 bezeichnet.

`REAL*8` ist die darstellung von "gleitkomma zahlen" in doppelter präsision (8 byte). In fortran wird dies als REAL*8 oder DOUBLE PRECISION bezeichnet.

Von **strings** in fortran-77 ist durch die funktion LEN lediglich die deklarierte länge bekannt:

```
      CHARACTER*8      A/'abc'/, B/'def'/
      CHARACTER*16     C
      ....
      C = A // B
      WRITE (*, 900) C
900   FORMAT ('>', A16, '<')
```

Die ausgabe durch das `WRITE` ergibt `>abc  def  <` und nicht `>abcdef  <`.

Deshalb wurde im precompiler McStrufo die möglichkeit gegeben, initialisierte strings mit einer impliziten längenangabe zu versehen. Diese kann durch die funktion Strlen gefunden werden. Es gilt immer Strlen(string) = 0...LEN(string). Dies wird durch einfügen eines speziellen zeichens in den string erreicht (I$DUMY).

1.2.1 Namenskonventionen

Namen von variablen, subroutinen und funktionen, die im kern des OBRZ runtime systems zu fortran verwendet werden, sollen gewissen regeln gehorchen. Dies ist insbesondere für globale namen (`COMMON, SUBROUTINE, FUNCTION`) notwendig.

`C$xxxx` namen von `COMMON` blöcken

`I$xxxx` namen von `INTEGER` werten, welche globale bedeutung haben, also zb in `COMMON` blöcken des runtime kernes vorkommen.

`R$xxxx` namen von `REAL` werten, welche globale bedeutung haben, also zb in `COMMON` blöken des runtime kernes vorkommen.

`$xxxxx` namen von werten irgendwelchen types, welche globale bedeutung haben, also zb in `COMMON` blöcken des runtime kernes vorkommen.

`yyyyy$` namen von subroutinen und funktionen, die im runtime kern vorkommen, aber nicht durch den benützer angewendet werden sollen. Dies sind zb routinen, welche den precompiler unterstützen.

1.3 Skelett der routinen

Allgemein verwendbare routinen - insbesondere jene aus der OBRZ runtime bibliothek - sollen folgendermassen aufgebaut sein:

```
subroutine/function NAME (INPUT-PARAMETER, OUTPUT-PARAMETER)

deklarationen: COMMON, type etc.

%INCLUDE (C$MSG)

TRACE= TRCSET ('NAME')
DEBUG= DBGSET ('NAME')
HELP = HLPSET ('NAME')    ! dies wird nur in sehr umfangreichen routinen
                          ! nötig, am ehesten im applikations programm
                          ! selbst.
! Da die meldungen im message file nacheinander stehen, sollen sie im
! programm auch entsprechend ihrer position durchnumeriert werden.

$nest = $nest + 1       ! increment nesting level

if TRACE then
   ausdrucken der input-parameter mit hilfe von MESSAG
endif

   Körper der routine bzw funktion:     Bearbeiten
   der eingangsparameter. Wenn wesentliche
   zwischenergebnisse entstehen, sollen diese für
   debugging gemeldet werden durch:

if DEBUG then
   ausdrucken der zwischen-ergebnisse mit  hilfe von MESSAG
endif

   Wenn im verlauf der bearbeitung fehlerbedingungen
   auftreten, so sollen die fehlermeldungen
   folgendermassen behandelt werden:

if FEHLERBEDINGUNG then
   VAL1 ... VAL5 in zeichenform bereitstellen
   call messag ('NAME', NR, IRETCD, ERRPOS, VAL1, VAL2 ... VAL5)
   if IRETCD .eq. 0 then
      alles ok, also was ist die default aktion?
   elseif IRETCD .eq. 4 then
      was muss getan werden?
   elseif IRETCD .eq. 8 then
      na wat denn?
   else
      default aktion
   endif
endif
```

```
   Vor dem verlassen der routine sollen alle
   zurückgegebnen werte - die output paramter
   (bei einer funktion: der funktionswert) -
   dargestellt werden:

if TRACE then
   ausdrucken der output-parameter mit hilfe von MESSAG
endif

$nest = $nest - 1       ! reset nesting level

RETURN
END
```

2 Funktionen für Bibliotheksroutinen

In diesem abschnitt wird beschrieben, wie die definierten mechanismen in den routinen der runtime bibliothek eingesetzt werden. In gleicher weise können sie natürlich auch von applikationen gebraucht werden.

Trace

Wenn eine program unit (subroutine, funktion) input von files liest, und TRACE ist eingeschaltet, dann soll der input als **echo** ausgegeben werden (mit hilfe der routine MSGPRT).

2.1.1 Funktion Trcset

Die LOGICAL*4 funktion

```
Trcset (namenliste)
```

liefert das ergebnis TRUE, wenn mindestens einer der in *namenliste* angegebenen namen als zu tracen ausgewiesen ist. *namenliste* ist dabei eine aufzählung der namen jener routinen, deren trace-funktion zu überprüfen ist. Die namen werden in apostrophen (als string) geschrieben.

Im allgemeinen wird diese funktion in der zu tracenden routine eingesetzt. Dann enthält *namenliste* nur den string mit dem namen der zu tracenden routine. Die funktion liefert also die antwort auf die frage, ob die momentane routine zu tracen ist oder nicht.

Soll der trace aller routinen abgefragt werden, so wird kein parameter angegeben. Die folgenden beispiele mögen dies verdeutlichen:

```
IF Trcset ('AXISVL') ...
IF Trcset () ...
```

Das erste beispiel fragt danach, ob für die routine AXISVL ein trace verlangt ist. Das zweite beispiel fragt danach, ob für irgend eine routine ein trace verlangt wurde.

2.2 Debug

2.2.1 Funktion Dbgset

Die LOGICAL*4 funktion

```
Dbgset (namenliste)
```

liefert das ergebnis TRUE, wenn mindestens eine routine, deren name in *namenliste* angegeben ist, debug output liefern soll. Die namen werden in apostrophen (als string) geschrieben.

Im allgemeinen wird diese funktion in jener routine eingesetzt, welche den debug output erzeugen soll. Dann enthält *namenliste* nur den string mit dem namen der momentanen. Die funktion liefert also die antwort auf die frage, ob die momentane routine zu debug output erstellen soll oder nicht

Soll der debug output aller routinen abgefragt werden, so wird kein parameter angegeben. Die folgenden beispiele mögen dies verdeutlichen:

```
IF Dbgset ('AXISVL  ') ...
IF Dbgset () ...
```

Das erste beispiel fragt danach, ob für die routine AXISVL ein debug output verlangt ist. Das zweite beispiel fragt danach, ob für irgend eine routine ein debug output verlangt wurde.

Meldungen

2.3.1 Routine Msgwrt

Die routine Msgwrt ist die allgemeine message routine. Sie stellt irgendwelche meldungen (trace meldungen, debug meldungen, fehler meldungen) dar. Der grösste teil eines meldungstextes ist auf einem file festgehalten. Lediglich variable werte (paramter) werden vor der ausgabe in diesen text eingepasst.

Msgwrt liest die texte zu den meldungen von einem file und stellt die meldung mit den aktuellen werten zusammen. Die fertige meldung kann auf verschiedene art ausgeben werden. Der aufruf lautet:

```
CALL Msgwrt (msgid, msgnr, iretcd, errpos,  p1 ... p5)
```

Die parameter haben dabei folgende bedeutung:

msgid String: enthält die identifizierung der meldung. Dies ist im allgemeinen der name jener routine, welche Msgwrt aufruft. Nur die ersten 8 stellen können verarbeitet werden. Für buchstaben sind kapitalien (grossbuchstaben) notwendig.

msgnr INTEGER. Nummer der meldung innerhalb der zur einem msgid gehörenden meldungen.

iretcd INTEGER. Diese variable erhält das 'gewicht' der meldung zurück.

Die folgenden parameter müssen nicht vorkommen. Von hinten her können angaben fehlen.

errpos INTEGER, gibt die position eines fehlers im string *p1* an. Diese position wird in der meldung durch eine markierung (unterstreichen, fettdruck, zeiger) hervorgehoben. Dieser parameter kann fehlen (hat den wert I$DUMY) oder 0 sein, wenn keine markierung von *p1* notwendig ist.

p1..p5 sind parameter zur meldung. Im meldungstext werden sie symbolisch mit #1 bis #5 angesprochen. Diese parameter sind strings. Numerische werte müssen also durch Mchr..(...) bzw `WRITE (AREA,...` in eine zeichenkette umgewandelt werden.

Die routine Msgwrt kann die anzahl der parameter mit hilfe von Nargum bestimmen. Deshalb können nicht gebrauchte p_i von hinten her weggelassen werden.

2.3.2 Routine Messag

Wem die methode der variablen parameterliste von Msgwrt nicht geheuer ist, für den stellt Messag eine Fortran-77 konforme methode bereit.

```
CALL Messag (msgid, msgnr, iretcd, errpos,  ntpar, text )
```

Die parameter haben dabei folgende bedeutung:

msgid String: enthält die identifizierung der meldung. Dies ist im allgemeinen der name jener routine, welche Messag aufruft. Nur die ersten 8 stellen können verarbeitet werden. Für buchstaben sind kapitalien (grossbuchstaben) notwendig.

msgnr INTEGER. Nummer der meldung innerhalb der zur einem msgid gehörenden meldungen.

iretcd INTEGER. Diese variable erhält das 'gewicht' der meldung zurück.

errpos INTEGER, gibt die position eines fehlers im string *p1* an. Diese position wird in der meldung durch eine markierung (unterstreichen, fettdruck, zeiger) hervorgehoben. Dieser parameter kann fehlen (hat den wert I$DUMY) oder 0 sein, wenn keine markierung von *p1* notwendig ist.

ntpar INTEGER, gibt die anzahl elemente im text-array *text* an.

text CHARACTER array mit *ntpar* elementen. In den aus der text-bibliothek kopierten meldungen werden diese aktuellen text-stücke symbolisch mit #1 bis #5 angesprochen. Numerische werte müssen natürlich durch Mchr..(...) bzw `WRITE (AREA,...` zuerst in eine zeichenkette umgewandelt werden.

Diese routine weist folgende begrenzungen auf:

- Aus der text-bibliothek können maximal 20 paragraphen verarbeitet werden (inlusive help-text).
- Ein paragraph darf nicht mehr als 1024 zeichen umfassen, sonst wird er abgeschnitten. Diese grenze gilt nach der substitution von symbolischen angaben #1 ... #n.
- Die zeilenlänge in den members mit den texten darf 256 zeichen nicht übersteigen.
- Sollblank (in Msgprt ersetzt. (_) underscore
- Bei fatalem fehler erfolgt kein abbruch.

2.3.3 Parameter zu meldungen

Da die meldungen in verschiedenen sprachen abgefasst werden können, sind die parameter symbolisch bezeichnet mit #1 bis #5. Sie können im meldungstext in beliebiger reihenfolge, auch mehrmals verwendet werden.

Die parameter werden als strings (zeichenketten) in den meldungstext eingefügt. Numerische werte sind also vor aufruf von Messag entsprechend umzuformen.

```
LENGTH= 28
 ...
CHARACTER*7 IMAGE
IF LENGTH > 16 | LENGTH < 1 <<
  WRITE (IMAGE, '(I7)')  LENGTH
  CALL Messag ('DPACKD', 2, IRETCD, 0, IMAGE)
  IF IRETCD > 0 <<
    DPACKD= $8DUMY
    RETURN
  >>
>>
```

2.3.4 Gewicht von fehlern

Fehler können unterschiedlich schwer sein. Dieses gewicht wird im meldungstext festgelegt - und nicht in der routine, welche den fehler feststellt. Dadurch kann das gewicht des fehlers je nach applikation verschieden sein. Das gewicht des fehlers ist damit von der umgebung abhängig.

Das aufgrund des meldungstextes ermittelte fehlergewicht wird dem parameter `IRETCD` zugewiesen. Im meldungstext wird das fehlergewicht durch einen grossbuchstaben angegeben:

I — **information** - gewicht 0. Diese meldungen sind für allgemeine informationen im rahmen von diagnose (trace, debug) etc. gedacht.

Im falle von fehlern wird dieses gewicht nur dann verwendet, wenn die routine entweder keine korrigierende aktion ausführen muss, oder die korrigierende aktion erfolgt nach anerkannten prinzipien (zb. werde anstelle von "exponent underflow" mit ergebnis 0 weiter gearbeitet).

E — **error** - gewicht 4. Für diese fehler ist eine korrigierende aktion in der routine mehr oder weniger sinnvoll. Wenn zb der logarithmus aus einer negativen zahl zu ziehen ist, werde mit dem betrag (absolutwert) weiter gearbeitet. Wenn ein string bei der übertragung in einen bereich nicht vollständig platz findet, liegt zb ein fehler dieses gewichts vor. Die übertragung wird einfach abgebrochen.

S — **severe error** - gewicht 8. Eine sinnvolle aktion zur korrektur des fehlers kann von der routine im allgemeinen nicht vorgenommen werden. Dies kann schon eher im aufrufenden programm bzw der applikation geschehen. Die routine wird das ergebnis auf **undefiniert** setzen. Ein fehlerhalfter string werde leer.

F — **fatal error** - gewicht 12. Ein fataler fehler verhindert im allgemeinen, dass die routine überhaupt sinnvoll arbeiten kann. Wenn z.b. zu wenig platz in einem pufferspeicher zur verfügung gestellt wird, liegt ein solcher fehler vor. Ein abbruch der routine und oftmals der gesamten applikation ist angezeigt. Im allgemeinen kann nur die applikation eine vage fehlerursache angeben.

A — **aborting error** - gewicht 16. Die routine und damit die applikation kann nicht fortgesetzt werden. Die fehlerdiagnose kann ohne eine analyse des specherinhalts nicht erfolgen. Deshalb wird ein **ABEND 4016** verursacht.

2.4 Variable Parameterliste

Die routinen der runtime bibliothek sollen so ausgerüstet werden, dass sie nur mit den notwendigen parametern aufgerufen werden müssen. Das heisst, nicht gebrauchtes wird ausgelassen oder (hinten) weggelassen.

Weggelassene parameter werden durch zählen der aktuellen parameter erkannt. Ausgelassene parameter werden an ihrem wert (=undefiniert) erkannt.

2.4.1 Funktion Nargum

Funktion Nargum gehört zum kern der runtime unterstützung. Sie ist in einem eigenen abschnitt detailliert beschreiben.

Diese INTEGER funktion kann ohne parameter aufgerufen werden und liefert die zahl der aktuellen argumente zur momentanen routine:

```
CALL XYZ (A, B, C)                SUBROUTINE XYZ (ALFA, BETA, GAMMA)
                                  NP= Nargum ()
CALL XYZ (Z)

CALL XYZ
```

Im ersten aufruf von XYZ wird Nargum den wert 3, im zweiten aufruf den wert 1 liefern und im dritten den wert 0. Für die gezeigte routine XYZ seien aber zb nur werte 1..3 gültig.

2.4.2 Funktion des Precompilers McStrufo

Variable argumentlisten werden durch den fortran precompiler McStrufo unterstützt, indem fehlende parameter durch den wert $4DUMY ersetzt werden:

```
CALL XYZ (A ,, C)                 CALL XYZ (A, $4DUMY, C)
CALL XYZ ()                       CALL XYZ ($4DUMY)
CALL XYZ (,,C)                    CALL XYZ ($4DUMY, $4DUMY, C)
CALL XYZ (A,B,)                   CALL XYZ (A, B, $4DUMY)
IF Nargum() ...                   IF Nargum($4DUMY) ...
```

Die variable $4DUMY hat den "wert" undefiniert.

2.4.3 Lange Parameterlisten

Um sehr lange parameterlisten oder unbekannte parameter-typen in den routinen der runtime bibliothek zu behandeln, wurde die routine **Getprm** geschaffen. Sie ist insbesondere dann hilfreich, wenn alle parameter von derselben art (zb lauter strings der länge 8) sind.

Diese routine gehört zum kern der runtime unterstützung und ist in einem eigenen abschnitt detailliert beschrieben.

```
CALL Getprm (iparm, area, length)
```

Die verwendung sei an einem trivialen beispiel gezeigt:

Aufrufendes programm

```
XYZ= SUM (A, B, C, D, E, F)
```

Aufgerufenes programm

```
FUNCTION  SUM (P)
REAL*8 VALUE, SUM
NP= Nargum()
SUM = 0.
DO J= 1, NP <<
  CALL Getprm (J, VALUE, 8)
  SUM= SUM+ VALUE
>>
```

2.5 Undefinierte Werte

Wenn auf fehlende daten geprüft wird, soll der wert $4DUMY bzw I$DUMY dazu verwendet werden. Diese sind im COMMON C$RUN enthalten. Die interne darstellung ist `X'FEFEFEFE'` und damit identisch zu den undefinierten werten in **Pascal** 8000.

Bei der beschreibung der funktionen bzw routinen ist angegeben, ob sie undefinierte daten berücksichtigen. Dies ist zb der fall für

`Double`	transformation von single in double precision
`Aprox`	polynomapproximation nach der methode der kleinsten quadrate

3 Trace back

Das FORTRAN VS runtime system liefert standardmässig einen sehr ausführlichen "trace back" - - eine liste der aufrufhierarchie der routinen. Bei jeder routine werden ihre parameter in INTEGER, REAL, CHARACTER und hexadezimal angegeben. Diese ausfürhlichkeit ist nicht immer erforderlich. Deshalb rufen die OBRZ routinen für den runtime support eine eigene routine (Errtr$) auf. Deren output wird über Msgprt erstellt.

Im gegensatz dazu kann die routine **Errtra nur** auf das "fortran error dataset" schreiben - im allgemeinen auf das file `FT06F001`.

Der output von Errtr$ sieht etwa so aus:

```
ERRTR$-00-I OBRZ Runtime Support Trace Back * * * * * RETURN-A   ENTRY-PT   FU  >
ROUTINE ...   WAS CALLED FROM ...          AT ISN     REG.  14   REG.  15   RE  >

IBCOM#        SKARD                                   0013EC50   001464B4   00
SKARD         PASS1                         0032      62139CC4   0013E8B0   00  >
PASS1         UT#TID                        0020      42138B9E   00138CB0   00  >
UT#TID                                                40126ED8   00138888   00  >
                                          ENTRY POINT ADDRESS = 00138888
```

3 Implementierung

In diesem abschnitt werden die interna der mechanismen beschrieben.

Die routine INITR$ muss nur dann zur initialisierung aufgerufen werden, wenn nicht mit der procedur FTN77 gearbeitet wird.

3 COMMON Blöcke

3.1.1 COMMON Block C$TRC

```
COMMON /C$TRC/ TRCIDS($NMAX)
```

TRCIDS ist als CHARACTER*8 deklariert. $NMAX sollte mindestens 50 sein.

3.1.2 COMMON Block C$DBG

```
COMMON /C$DBG/ DBGIDS($NMAX)
```

DBGIDS ist als CHARACTER*8 deklariert. $NMAX sollte mindestens 50 sein.

3.1.3 COMMON Block C$HLP

```
COMMON /C$HLP/ HLPIDS($NMAX)
```

HLPIDS ist als CHARACTER*8 deklariert. $NMAX sollte mindestens 50 sein.

3.1.4 COMMON Block C$MSG

In diesem COMMON block sind sehr viele werte definiert, deren reihenfolge mit der liste übereinstimmen:

$MSGIN LOGICAL. Wird **false**, sobald das message-system initialisiert ist.

$MXLEV INTEGER. Hält das höchste gewicht aufgetretener fehler fest.

$OFILE INTEGER. Bestimmt die art der ausgabe der meldungen. Siehe routine "MSGOPT".

$MXPRT INTEGER. Gibt an, wie oft meldungen gedruckt werden sollen. Siehe routine "MSGOPT".

$LINLN INTEGER. Zeilenlänge für die druckausgabe der meldungen. Default ist 132.

$LINCT INTEGER. Anzahl verfügbarer zeilen auf der druckseite (gesamte seite).

$LINIS INTEGER. Gibt die noich verfügbaren zeilen auf der druckseite an. Dieser wert wird nach dem drucken einer zeile um 1 vermindert. Wenn dann auf 0 stehend, wird er wieder auf $LINCT gesetzt und auf eine neue seite gesprungen.

$MSGLV INTEGER. Gibt an, wie ausführlich die meldungen sein sollen. Siehe routine "SETMSG".

$ISERR LOGICAL. Wird auf true gesetzt, sobald MESSAG bzw MSGWRT das erste mal eine meldung mit einem gewicht > 0 erstellen musste.

$NEST INTEGER. Hält die schachtelungstiefe der routinen fest. Dabei stellt das applikations (haupt-)programm die ebene 0, die routine MESSAG bzw MSGWRT die "tiefste" ebene dar.

$APPL String der länge 32 mit dem namen der applikation (oder leer). Die ersten 8 zeichen geben dabei einen kurznamen (in grossbuchstaben) an, der für die referenz zu den members mit den texten verwendet wird.

$NMAX INTEGER. Dieser wert muss mit der DIMENSION angabe von TRCIDS, DBGIDS und HLPIDS übereinstimmen. erwünscht wird. Siehe routinen SETTRC, CLRTRC und TRCSET.

$TRCDF LOGICAL. Gibt an, ob alle namen in TRCIDS definiert sind.

$DBGDF LOGICAL. Gibt an, ob alle namen in DBGIDS definiert sind.

$HLPDF LOGICAL. Gibt an, ob alle namen in HLPIDS definiert sind.

3.1.5 COMMON Block C$RUN

In diesem bereich werde werte gehalten, die von allen runtime routinen des OBRZ systems verwendet werden. Dies sind insbesondere solche zur unterstützung des precompiler McStrufo.

```
COMMON /C$RUN/ $8DUMY, $4DUMY, I$DUMY, I$ESTR, ICPIN, ICPOUT, $SPARE(25)
REAL*8          $8DUMY
```

$8DUMY REAL*8: "undefiniert"

$4DUMY REAL*4: "undefiniert"

I$DUMY INTEGER: "undefiniert"

I$ESTR INTEGER: nummer des zeichens für string-termination (0..255). Zum gebrauch in CHARACTER daten muss es mit CHAR(I$ESTR) eingesetzt werden. Dieser umweg ist notwendig, weil character type nicht mit anderen types in COMMON gemischt werden darf.

ICPIN INTEGER: gibt die nummer der codepage für input an. Default ist 37 (US-EBCDIC)

ICPOUT INTEGER: gibt die nummer der codepage für output an. Default ist 37 (US-EBCDIC)

$SPARE reserve (25 integer)

I$EARR CHARACTER * 1024

3.1.6 COMMON Block C$AREA

In diesen common block werden die texte der meldungen geschrieben, wenn die variable $OFILE < 0 ist.

$MSGLN INTEGER, Dieser wert gibt die länge des nachfolgenden bereiches in zeichen an. Er sollte mindestens 80 stellen umfassen, um wenigstens den beginn einer meldung sinnvoll wiederzugeben. Die meldungen werden für einen bereich von 512 zeichen ausgelegt - ein kurzer bereich bringt also unnötige einschränkungen.

$MSGTX ist ein bereich mit der deklaration CHARACTER**$MSGLN* Die tatsächliche länge des meldungstextes wird markiert, wenn die ganze meldung platz finden kann.

3.1.7 Verwendung der COMMON Blöcke

COMMON	routinen, die ihn verwenden
C$AREA	MESSAG, MSGWRT, applikation
C$DBG	SETDBG, CLRDBG, DBGSET, MESSAG, MSGWRT
C$MSG	SETMSG, SETHLP, CLRHLP, MSGOPT, MESSAG, MSGWRT
C$RUN	alle routinen der runtime bibliothek
C$TRC	SETTRC, CLRTRC, TRCSET, MESSAG, MSGWRT

2 Festlegungen

3.2.1 Standardwerte

Bei der initialisierung werden folgende werte gesetzt:

variable	default wert
$DBGDF	false
$HLPDF	false
$LINCT	66
$LINIS	0
$LINLN	132
$MSGLN	256
$MSGLV	3
$MXPRT	50
$NEST	0
$OFILE	6
$TRCDF	false

3.2.2 Bereiche und Konstanten

Bereiche und werte in den programmen für die handhabung der meldungen sind in der folgenden tabelle zusammengestellt.

name	zweck des bereiches	wert
lrecl	maximale recordlänge für die message=files	256
txtmax	maximale länge des rohen meldungstextes (vor dem ausscheiden überflüssiger leerstellen)	1024
$NMAX	maximale zahl verschiedener routinen für individuellen TRACE, DEBUG oder HELP (arrays TRCIDS, DBGIDS, HLPIDS)	50
$4DUMY	undefinierter REAL*4 wert	x'FEFEFEFE'
$8DUMY	undefinierter REAL*8 wert	x'FEFEFEFE..'
I$DUMY	undefinierter INTEGER wert	x'FEFEFEFE'
I$ESTR	"character" zur kennzeichnung eines string-endes	254
I$EARR	text-bereichzum zusammenstellen von meldungen	1024 char

3.3 Meldungen

3.3.1 Funktion der Routinen MESSAG und MSGWRT

Die routinen MESSAG und MSGWRT erfüllen folgende funktionen:

- prüfen der aktuellen argumente zu MESSAG bzw MSGWRT
- decodieren der meldungsnummer *msgnr*
- suchen des members mit den meldungen in der entsprechenden bibliothek mit hilfe von GETLN$. Wenn nicht zu finden: meldung und abbruch von MESSAG bzw MSGWRT
- suchen der meldung im member gemäss *msgnr*. Wenn nicht zu finden: meldung und abbruch von MESSAG bzw MSGWRT
- lesen der meldung in den puffer
- ersatz der symbolischen werte in der meldung durch die aktuellen parameter von MESSAGbzw MSGWRT. Sollen die aktuellen werte in besondere zeichen eingeschlossen werden, muss der symbolische wert im message-file mit diesen zeichen eingeklammert werden. Leerstellen in den werte-strings werden als unterstreichzeichen eingebracht, damit sie nicht verloren gehen.
- ausgeben der meldung:
  - auf den drucker durch MSGPRT
  - auf den job-log durch die routine SBS002 (process-indikator entsprechend aufsetzen).
  - direkt in den common block C$AREA, aus dem die applikation dann die meldung entnehmen und weiter verarbeiten kann.
- im text der meldungen sind folgende speziellen zeichen zu beachten:

  `&` wird für variablen des textformatters SUSI gebraucht, und muss daher im meldungsfile als && geschrieben werden. MESSAG bzw MSGWRT muss aus je zwei zeichen eines machen.

  `$` hat bedeutung in SUSI und muss daher doppelt geschrieben werden ($$). MESSAG bzw MSGWRT muss aus je zwei zeichen eines machen.

  `|` mögliche trennstelle im text. Wenn als zeichen | gebraucht, muss es daher im meldungstext doppelt geschrieben werden (||). MESSAG bzw MSGWRT muss aus je zwei zeichen eines machen.

  `_` (sollblank) hält zeichenfolgen zusammen. MESSAG bzw MSGWRT muss aus diesem zeichen leerstellen machen (für output durchSBS002 oder in den COMMON bereich $MSGTX).

  `-` (minus) auf position 5 forciert eine neue output zeile (liste)

  `{}` sollen im text dazu benutzt werden, symbolische parameter einzuschliessen. Für MESSAG bzw MSGWRT haben diese zeichen keine besondere bedeutung. Sie können aber vom formatter *SUSI* zu einem highlight (kursiv) verwendet werden.

- falls erwünscht: lesen und drucken des zur meldung gehörenden "help"-textes.
- setzen von *iretcd* aufgrund des meldungs-gewichtes
- setzen weiterer indikatoren in den COMMON blöcken
- abbruch der applikation bei fatalem fehler
-

Beim drucken des meldungstextes wird der text mit einer einrückung gemäss $NEST versehen. Dadurch kann die aufrufhierarchie leicht wahrgenommen werden - solange nicht anderer output

dazwischen kommt. Insbesondere kennt die routine ERRTRA keine einrückung (liesse sich aber machen).

Der erste aktuelle wert in der meldung kann besonders gekennzeichnet werden: mit dem parameter *errpos* wird jene stelle im string *p1* angegeben, an welcher der fehler festgestellt wurde. *errpos* ist 0, wenn keine kennzeichnung notwendig ist. Für die ausgabe dieser zeilen muss MSGPRT mit *ifunct* < 0 aufgerufen werden.

3.3.2 Routine GETLN$

Diese routine liest mit hilfe der CLUE routine URRPDS eine zeile aus dem message file. Dabei werden bestimmte zeilen (markierte und kommentar) immer übersprungen. Das message file wird immer unter dem DD-namen RUNMSGS erwartet.

```
CALL GETLN$ (member, area, atype, etyp, length, cont, errgew)
```

Dabei haben die parameter folgende bedeutung:

member String: name des members im message file, aus dem die texte zu lesen sind.

area String: nimmt die gelesene zeile auf. Die tatsächliche länge ist mit I$ESTR markiert.

atyp String: gibt an, in welcher zeilengruppe das lesen begonnen werden soll. Zb 'DS' oder 'MS' oder 'MT02'. MT02 bedeutet dabei die gruppe "meldungstext 02".

etyp String: gibt an, bei welcher zeilengruppe mit dem lesen aufgehört werden soll. Zb 'PR' oder 'HE'.

length INTEGER: positive werte geben an, wie lange die gefüllte zeile *area* ist. Negative werte geben fehlerbedingungen an:

- -2 gesuchtes *atyp* kann nicht gefunden werden. Somit kann der gewünschte text nicht bereitgestellt werden.
- -n wobei n> 7: Dies sind fehler-angaben, die direkt aus der routine URRPDS übernommen werden. In einem solchen fall muss MESSAG bzw MSGWRT eine geeignete meldung via SBS002 in den joblog absetzen (dieser code muss darin vorkommen).

cont (logical) gibt an, ob noch weitere zeilen folgen.

errgew gewicht der (fehler)-meldung.

GETLN$ liest nur eine zeile, damit MESSAG bzw MSGWRT möglichst freie hand hat und zb die symbolischen werte zu beginn der zeilen leicht findet.

Der für MESSAG bzw MSGWRT relevante teil der zeilen beginnt erst auf position 5 (siehe file beschreibung). Jene zeilen, welche mit $atype beginnen werden nur dann an MESSAG bzw MSGWRT übergeben, wenn sie tatsächlich etwas enthalten (1. zeile einer meldung). Auch die kommentarzeilen werden übersprungen.

Der apostroph nach der meldungsnummer in der ersten zeile eines meldungstextes wird durch GETLN$ entfernt. Er ist für die bearbeitung der texte durch den textformatter SUSI notwendig (listen konstrukt).

3.3.3 Routine MSGPRT

Diese routine arbeitet den meldungstext für die darstellung auf dem drucker auf:

```
CALL MSGPRT (ifunct, indent, text, markit)
```

Dabei haben die parameter folgende bedeutung:

ifunct INTEGER: gibt an, was zu machen ist:

<0	eine zeile wird ab *indent* eingerückt ausgegeben. Falls kein platz mehr, wird sie abgeschnitten Diese funktion ist für **echo** von input zu routinen gedacht und auch zum ausgeben der tatsächlichen meldung, die von MESSAG bzw MSGWRT vollständig aufbereitet werden muss. Dies insbesondere dann, wenn der fehlerort angezeigt wird. Insbesondere weist die erste ausgabezeile eine andere (kleinere) einrückung auf als die folgezeilen.
=0	????
>0	der text ist auf zeilen zu verteilen, die gemäss $NEST eingerückt werden. Dabei werden die zeilen an geeigneten stellen (blanks) gebrochen. Diese funktion wird dann eingesetzt, wenn MESSAG bzw MSGWRT möglicht entlastet werden soll: ausgeben von help-text, beschreibung etc.

indent INTEGER: anzahl zeichen, die eingerückt werden soll (innerhalb von $LINLN).

text String mit dem darzustellenden text. Dabei ist die länge durch ein spezielles zeichen (I$ESTR) markiert. Der *text* kann noch unterstreichzeichen enthalten. Diese werden erst nach dem aufteilen in zeilen in leerstellen umgewandelt.

markit INTEGER: position im text, die speziell markiert werden soll (zb durch unterstreichen).

Für die darstellung wird später ein beispiel gegeben.

3.3.4 Routine INITR$

Diese routine füllt die COMMON bereiche des OBRZ runtime systems mit den notwednigen werten. Ein aufruf dieser routine (ohne parameter) ist nur notwendig, wenn nicht die mechanismen des OBRZ fortran systems verwendet werden können.

Normalerweise wird beim linken ein BLOCK DATA zugefügt, welches diese daten definiert.

4 Bibliothek der Meldungen

Parallel zu den bibliotheken mit den ausfürbaren modulen der OBRZ runtime bibliotheken werden solche mit den texten zu den meldungen eingerichtet.

Für jede routine besteht in dieser bibliothek mindestens ein member, das den zugehörigen text enthält.

Wenn die routinen der runtime bibliothek im rahmen einer applikation eingesetzt werden, kann eine weitere bibliothek mit applikationsbezogeneren meldungen bestehen. Dieses wird **vor** das allgemeine file concatiniert. Dadurch wird zuerst die applikationsbezogene meldung gefunden.

Der aufbau dieser bibliotheken ist identisch. Sie sollen die attribute

```
RECFM= VB, LRECL < 257
```

aufweisen, damit sie leicht editiert werden können. Der inhalt der members wird später beschrieben. Er ist so gestaltet, dass die texte auch mit einem formatter (SUSI) bearbeitet werden können - für eine ansprechende dokumentation.

Das file mit meldungen soll immer parallel zur entwicklung der routine entstehen.

3.4.1 Namensgebung

Die namen der members in der bibliothek mit den meldungen müssen wie folgt gebildet werden:

- Zunächst einmal stimmt der membername mit dem namen der routine überein, zu welcher die meldungen gelten.
- wenn in der routine (oder insbesondere der applikation) mehr als 99 meldungsnummern notwending werden, wird in gruppen unterteilt:
  - für bis 999 meldungsnummern wird der name der members auf 7 stellen gekürzt und eine numerierung (0..9) angehängt.
  - für bis 9999 meldungsnummern wird der name der members auf 6 stellen gekürzt und eine numerierung (00..99) angehängt.

Solange also weniger als 100 meldungen zu einer routine gehören (und 10 sind schon recht viel), sind diese alle im selben member gesammelt.

Wenn die meldungen aus einer applikation (sie habe zb den namen APPLIKAT) in 4 gruppen aufgeteilt werden können, so werden vier members mit meldungen erstellt: APPLIKA0, APPLIKA1, APPLIKA2 UND APPLIKA3 (sie können auch anderen end-ziffern haben, wie die weitere erklärung zeigt).

In diesen members müssen nicht 99 meldungen stehen, was bei gruppenbildung auch nicht sinnvoll ist. Aber die meldungen in den members müssen mit der nummer im membernamen übereinstimmen, also:

- im member APPLIKA0 stehen alle meldungen 001 ... 099
- im member APPLIKA3 stehen alle mit den nummern 301 ... 399.

Diese übereinstimmung ist notwendig, da zuerst das entsprechende member ausgesucht und dann darin die meldungen gesucht werden.

Es ist nicht notwendig, dass die meldungen zu einer routine fortlaufend numeriert werden. Vielmehr kann von anfang an eine gruppierung vorgenommen werden etwa nach dem schema input - verarbeitung - output.

3.4.2 Inhalt der Members

Der meldungstext selbst muss "numerierte records" (ISPF jargon) verwenden, sonst stimmen die aufteilungen nicht. Nach der nummer, welche die record-positionen 1..8 belegt, folgt der eigentliche text mit dem nachstehend beschriebenen aufbau.

Die meldungen müssen entsprechend ihren nummern aufsteigen angeordnet werden, da nur sequentiell danach gesucht wird.

`9..12` steuerung für formatter und routine MESSAG bzw MSGWRT. Dies ist im wesentlichen eine identifizierung der textelemente (&CP&, &LB&, &NM& etc.)

`13..n` meldungstext, helptext etc. MESSAG und MSGWRT erhalten nur den text aus diesem bereich (text-postion 5 .. n).

Der text weist zunächst die elemente auf,die für die ganze routine gelten, dann folgen abschnitte der einzelnen meldungen, welche in die "unterabschnitte" meldungstext und helptext unterteilt sind. Dabei werden die helptexte noch in listen gegliedert.

Kommen im text auf den postionen 9..12 steuerangaben vor, die hier nicht erlätert sind, so haben diese für die routine MESSAG bzw MSGWRT keine bedeutung. Solche zeilen werden wie kommentarzeilen übergangen. Dadurchkönnen besondere angaben für den formatter eingebracht werden.

`.*` leitet eine zeile mit beliebigem kommentar ein. Solche zeilen können beliebig verwendet werden und dienen nur dem menschlichen leser.

`&CP&` ist unbedingt in solche texte einzufügen, die in **codepage 500** erstellt werden. Dies werden insbesondere die deutschen texte sein. Wenn diese angabe notwendig ist, **muss** sie auf der ersten zeile stehen.

`&P&` Zwischenraum in der liste. Damit können längere texte bewusst in absätze unterteilt werden. Dies wird besonders im beschreibungstext notwendig sein.

Die folgenden abschnitte werden sowohl von einem formatter, als auch von der routine MESSAG bzw MSGWRT behandelt. Als erstes muss dieser abschnitt eingeleitet werden mit:

`&MS&` hinter dieser angabe steht nichts.

Jede meldung erfordert zwei elemente, die eigentliche meldung und ein zusätzlicher "help"-text in form einer liste möglicher ursachen.

`&MT&` meldungstext, beginnend mit der nummer und dem fehlergewicht. Am deutlichsten zu sehen im beispiel.

Damit die meldungsnummer leicht gefunden werden kann, wird sie stets rechtsbündig in den positionen 13..16 eingetragen. Auf den positionen 17 und 18 folgt dann ein minus-zeichen und der buchstabe für das fehlergewicht. Nach einem apostrophen wird dann der text geschrieben.

Symbolische werte werden durch #n (wobei n 1..5 sein kann) angegeben. Diese symbolischen sind unter umständen in marken wie " " < > einzuschliessen. da MESSAG bzw MSGWRT keine hervorhebung einfügt.

Dieser text wird natürlich nur dann verarbeitet, wenn $MSGLV > 0 ist.

`&HE&` leitet eine liste möglicher ursachen zum fehler ein. Auf dieser zeile steht selbst kein text. Vielmehr werden die texte der folgenden zeilen als liste geführt: jedes neue listenelement wird durch einen punkt (.) in pos 9 und einen bindestrich (-) in pos. 13 eingleitet. Der text selbst steht nur auf pos 9..n.

Dieser text wird nur dann gedruckt, wenn $HELP gesetzt (true) ist.

`&EH&` schliesst die liste der möglichen erklärungen ab.

Die liste aller meldungen wird abgeschlossen durch:

&EM& dahinter steht nichts

Die teile, welche für die dokumentation der routinen wesentlich sind, folgen als letztes. Diese teile können recht umfangreich werden. Es wäre nicht sinnvoll, sie an den anfang zu stellen, weil dann bei der behandlung jedes fehlers darüber hinweg gelesen werden müsste. Beim erstellen der dokumentation mit dem formatter SUSI muss dann allerding die reihenfolge umgestellt werden. Dies geschieht durch zwei INCLUDES mit 'anfang, ende' selection. Eine andere methode wäre, den beschreibenden text in einer weiteren bibliothek unterzubringen. Aber wir haben ja schon sehr viele ...

&NM& name der routine, zu der die texte gehören. Auch dies ist nur für den formatter bestimmt.

&DS& beschreibung der routine. Sie soll so abgefasst sein, dass sie als programmbeschreibung sinn macht. Kürze ist hier nicht unbedingt angezeigt, sondern klarheit.

&PR& bescheibung der einzelnen parameter der routine in der form einer "definitions liste". Dazu wird für jeden parameter in pos 9 ein punkt (.) gesetzt und der name des parameters wird in apostrophen ab pos 13 gesetzt. Dahinter kommt (immer ab pos 13) die bescheibung des parameters.

&EP& schliesst die liste der parameterbeschreibung ab.

&LB& bezeichnung der bibliothek, diese angabe ist redundant und nur für den formatter bestimmt

&EX& Beginn eines beispiels

&EE& Ende eines beispiels

Das folgende ist ein vollständiges beispiel eines text-members. Es handelt sich um DPACKD aus der bibliothek TWLIB.F77.OBRE.TEXT:

```
.*====================================================================
&MS&
.*--------------------------------------------------------------------
&MT&  01-I'Trace at ENTRY: {field} (hex) = #01, {length} = #02.
&HE&
.    - {field} is the string to be transformed, it has a length of
       {length}.
&EH&
.*--------------------------------------------------------------------
&MT&  02-S'Length of string {length} (#01) is greater than 16 or lower
       than 1. The function value {Dpackd} is set to {undefined}.
       {length} must have an INTEGER value greater than zero and lower or
       equal than 16.
&HE&
.    - Decimal packed data may be up to 16 bytes long giving at most 31
       decimal figures plus a sign. Longer fields are not defined in the
       360/370 architecture.
.    - Check the application program which should restrict the reading
       of packed data to at most 16 bytes.
&EH&
.*--------------------------------------------------------------------
&MT&  03-S'{field} contains invalid elements; {field} (hex) = #01. The
       function value {Dpackd} is set to {undefined}.  Except of the last
       4 bits the packed data in {field} may only contain halfbytes of
       hex {0} to hex {9}.
&HE&
.    - The last halfbyte in {field} must have a value between hex {A} and
```

```
      hex {F}.
&EH&
.*-------------------------------------------------------------------
&MT&  04-S'The last halfbyte of {field} has invalid value: {field} (hex)
      = #01. The function value {Dpackd} is set to {undefined}. The last
      halfbyte in {field} must have a value between hex {A} and hex {F}.
&HE&
.   - This halfbyte contains the sign of the packed data; hex {A}, {C},
      {E}, {F} are positve numbers whereas hex {B} and hex {D} stand
      for negativ numbers.
&EH&
.*-------------------------------------------------------------------
&MT&  05-I'Trace at EXIT: {Dpackd} = #01.
&HE&
&EH&
.*-------------------------------------------------------------------
&EM&
.*===================================================================
&NM&Dpackd
&DS&
    DOUBLE PRECISION FUNCTION Dpackd converts packed data to REAL*8
    numbers. The packed data is given to Dpackd in the string {field}
    and the length of the string in {length}. The format of the packed
    data is {DD...DDS}: The D elements contain the decimal coded
    absolute value of the data and the S element is the sign. All
    elements are halfbytes (4 bits). The D elements have values of hex
    {9} and lower and the S element contains hex {A} or higher. Hex
    {A}, {C}, {E}, {F} are interpreted as positive signs and hex {B}
    and hex {D} as negativ signs.
&P&
    Error messages are generated when an invalid length is specified
    or when the data has a wrong format.
&P&
    When calling function Dpackd their parameters are:
&PR&
.   'field'  string containing the packed data
.   'length' the length
&EP&
&LB&TWLIB.F77.OBR
.*===================================================================
```

3.4.3 Meldungen zu Applikationen

Zu beginn der applikationsentwicklung entält die meldungsbibliothek dasselbe wie die meldungen zur runtime bibliothek der verwendeten routinen.

Die abschnitte &DS& ... &EP& beschreiben aber die einzelnen gruppen von meldungen.

Mit fortschreitender erfahrung in der interpretation von fehlern der routinen werden abschnitte &HE& - &EH& zu den meldungen immer spezifischer gestaltet. Das heisst, die möglichen erklärungen zu den fehlern beziehen sich immer stärker auf die konkrete situation der applikation (mit entsprechenden begriffen aus der applikation).

Regeln zur Bildung von Texten zu fehlermeldungen

Diese regeln wurden aus dem artikel von ... im IBM System Technical Journal .. 1982 kondensiert:

- formuliere positiv, nicht negativ:

  schlecht Sie haben einen falschen wert für xyz angegeben!

  besser Mögliche werte für xyz sind a, b oder c - der angegebene wert ist z. Es wird mit c weiter gearbeitet

- Mathematische abkürzungen und anderes vermeiden, wenn sie nicht in mathematischen routinen etc. verwendet werden:

  schlecht Die länge des 1. parameters muss 1 < l < 10 sein; l= 17.

  besser Der 1. parameter kann nur zwischen 2 bis 9 zeichen lang sein. Der angegebene wert ist 17.

- Die meldung muss aussagefähig und hilfreich sein:

  schlecht Der wert nparm=0 ist ungültig!

  besser Subroutine xyz wurde ohne parameter aufgerufen. Mindestens 1 parameter ist notwendig.

- Neutrale statt persönliche aussage:

  schlecht Sie haben vergessen, für routine xyz ...

  besser Für routine xyz ist mindestens 1 parameter notwendig.

3.5.1 Beispiel einer Meldung

Wenn das folgende beispiel ausgeführt wird

```
LENGTH= 28
 ...
CHARACTER*7 IMAGE
IF LENGTH > 16 | LENGTH < 1 <<
  WRITE (IMAGE, '(I7)')  LENGTH
  CALL MESSAG ('DPACKD', 2,IRETCD, 0, IMAGE)
  IF IRETCD > 0 <<
    DPACKD= $8DUMY; RETURN
  >>
>>
```

und das im abschnitt 3.4.2 dargestellte meldungsfile gilt, so entsteht folgende meldung. Dabei ist der parameter *LINGUA* in der JCL procedur FTN77 auf E gesetzt:

```
DPACKD-02-S Length of string {length} (28) is greater than 16 or lower
      than 1. The function value {Dpackd} is set to {undefined}.
      {length} must have an INTEGER value greater than zero and lower or
      equal than 16.
```

Der parameter IRETCD von MESSAG bzw MSGWRT erhält den wert 12. Falls "help" für diese routine verlangt ist, lautet die meldung:

```
DPACKD-02-S Length of string {length} (28) is greater than 16 or lower
      than 1. The function value {Dpackd} is set to {undefined}.
      {length} must have an INTEGER value greater than zero and lower or
      equal than 16.
      - Decimal packed data may be up to 16 bytes long giving at most 31
        decimal figures plus a sign. Longer fields are not defined in
        the 360/370 architecture.
      - Check the application program which should restrict the reading
        of packed data to at most 16 bytes.
```

In einer applikation, beispielsweise UT#GRAPH, kann dieser text aber so aussehen:

```
DPACKD-02-S Länge eines gepackten feldes ist grösser als 16 oder kleiner
      als 1. Der zu lesende wert wird auf UNDEFINIERT gesetzt.
      - Vermutlich ein fehler des programmierers (routine LESEN). Bitte
        OBRZ/DTA verständigen.
      - Gepackte daten können bis 16 bytes lang sein, was zu einer
        maximal 31 stellen langen dezimalzahl (plus vorzeichen) führt.
        Längere felder sind in der 360/370 architektur nicht definiert.
```

Wenn diese applikation von der trace möglichkeit gebrauch macht und der trace für die routine DPACKD eingeschaltet ist, so könnte sich diese wie folgt melden:

```
DPACKD-04-I TRACE at ENTRY: {field} (hex) = 00000003456C, {length} = 6.
```

ABSICHTLICH LEER GELASSEN

4 Programmierunterstützung

4.1 Bibliotheken des Fortran systems

Die erste gruppe von bibliotheken (mit * markiert) sind sogenannte "receive" datasets des SMP/E prozesses (target libraries).

TWLIB.F77.	COMP	LOAD	FORTRAN VS (V1) compiler *
	FORTLIB	LOAD	FORTRAN VS (V1) runtime library * (dynamisch geladene module)
	LINKLIB	LOAD	FORTRAN VS (V1) runtime library * (zur linkzeit eingebundene module)
	ALTLIB	LOAD	alternative math routinen *
	IAD	LOAD	Interactive debug zu FORTRAN VS *
	IAD	CLIST	CLISTS zu IAD *
	IAD	MSGS	Meldungen zu IAD *
	IAD	HELP	Help texte zu IAD *
	IAD	PANELS	ISPF panels zu IAD *
	IAD	SAMPLES	??? zu IAD *
	PROCLIB		Proceduren *
	SAMP	SOURCE	Beispiele und fehler-tests
TWLIB.F77.	PRE	SOURCE	Fortran precompiler McStrufo (neue version in Fortran 77)
	PRE	OBJ	
	PREE	TEXT	meldungstexte in englisch
	PRED	TEXT	meldungstexte in deutsch
TWLIB.F77.	RUN	SOURCE	OBRZ modifikationen / erweiterungen (zb alle routinen für 1recompiler support, message handling etc.)
	RUN	LOAD	
	RUN	TEST	
	RUNE	TEXT	meldungstexte in englisch
	RUND	TEXT	meldungstexte in deutsch
	OBR	SOURCE	Allgemein verwendbare routinen (mathematik, filehandling, etc.)
		LOAD	
		TEST	
	OBRE	TEXT	meldungstexte in englisch
	OBRD	TEXT	meldungstexte in deutsch
	PLT	SOURCE	Benützerseite des plot-systems
		LOAD	
		TEST	
	PLTE	TEXT	meldungstexte in englisch
	PLTD	TEXT	meldungstexte in deutsch
	OPT1	SOURCE	macros, description, DSECTS
	OPT2	SOURCE	modules

Im allgemeinen werden wir zuerst nur die englischen texte bereit stellen. Diese sind kürzer, erfordern keine umlaute, und eine mischung zwischen deutsch und computer jargon wird vermieden. Für applikationen sollten aber deutsche texte erstellt werden.

FORTRAN "house style OBRZ"

Programme in den OBRZ fortran bibliotheken sollen nach möglichkeit den folgenden regeln entsprechen. Sie sind anschliessend an einem beispiel gezeigt.

- COMMON blöcke und andere globale deklarationen werden über ein INCLUDE definiert. Dazu soll der precompiler verwendet werden, und nicht der compiler, damit auch die eingeschlossenen teile kleinbuchstaben und inline comment aufweisen können. Die form lautet

  ```
  % INCLUDE dataset (membername)
  ```

  Der datasaset-name wird (noch) nicht beachtet. Es wird also immer aus der selben bibliothek kopiert.
- Die funktionen des precompilers McStrufo werden so weit als möglich ausgeschöpft (inline kommentar, umsetzung von klein- auf grossbuchstaben, erweiterte steueranweisungen, unterstützung der string-handhabung, ...)
- Schlüsselwörter wie DO, WHILE, WRITE, FORMAT etc. werden in grosbuchstaben geschrieben.
- Namen von variablen werden in kleinbuchstaben geschrieben (j, nparm, ...)
- Namen von subroutinen und funktionen werden mit grossem anfangsbuchstaben geschrieben (Sin, Nargum, Input, Urrpds, ...)
- Der einleitende kommentar zu einer routine soll unmittelbar nach der subroutinen/funktionendeklaration angeordnet werden und nur die notwendigsten angaben (kurzbeschreibung der parameter) enthalten.
- Kommentare werden mit einem grossen C in der ersten postion oder einem ! in beliebiger position (vorzugsweise pos 41) gekennzeichnet.
- Die eigentliche beschreibung der routine wird im rahmen der help- texte in `TWLIB.F77...TEXT` anggeben. Siehe hierzu den abschnitt "inhalt der members".
- Der FortranVS compiler fordert, dass DATA nicht mit typendeklarationen gemischt werden dürfen. Deshalb müssen INCLUDE's zu COMMON blöcken etc. vor DATA angeordnet werden. Auch die anweisung RUNTIME SUPPORT, welche deklarationen generiert, muss vor den lokalen DATA anweisungen stehen.
- Für die "kern"-routinen des OBRZ runtime systems (`TWLIB.F77.RUN..`) werden für typ CHARACTER nur original Fortran-77 deklarationen verwendet. Der gebrauch des string terminators ist auf CHAR(I$ESTR) und die precompiler funktion STRING('zeichenkette') begrenzt. Daten-initialisierungen sollen deshalb für diesen typ gemäss beispiel erfolgen.
- Anweisungen, deren wirkung nicht offensichtlich ist, müssen mit "inline" kommentar beschrieben werden (kommentar steht nach einem !).
- Die routinen werden gemäss dem skelett in abschnitt 1.3 aufgebaut werden.

Das folgende beispiel (`DPACKD` aus `TWLIB.F77.OBR`) zeigt nicht alle der hier vermerkten regeln.

```
      DOUBLE PRECISION FUNCTION Dpackd (field, length)
C-----------------------------------------------------------------------+
C Zweck:                                                                |
C   uebertraegt gepackte Daten in REAL*8 Zahlen                         |
C                                                                       |
C History:                                                              |
C   K. Daube       3.85    Verfasser                                    |
C   T. Hunziker   12.86    FORTRAN 77, Meldungen                        |
C-----------------------------------------------------------------------+
        CHARACTER*(*) field
```

```
      REAL*8         value
      INTEGER        length, iretcd, ii, ival
      CHARACTER*32   dpastr, Mchrx, arg(2), Mchrr8
      CHARACTER*16   Mchri
      LOGICAL        trace, Trcset
%     INCLUDE (C$MSG)
%     INCLUDE (C$RUN)
      $nest = $nest + 1
      trace = Trcset ('DPACKD')
      IF (trace) THEN
        arg(1) = Mchrx (field, 0)
        arg(2) = Mchri (length)
        CALL Messag ('DPACKD', 1, 0, 0, 2, arg)
      ENDIF
      IF ((length.GT.16).OR.(length.LT.1)) THEN
        CALL Messag ('DPACKD', 2, iretcd, 0, 1, Mchri (length))
        Dpackd = $8dumy
      ELSE
        dpastr = Mchrx (field, 2 * length)
        value = 0.D0
        DO 100 ii = 1, 2 * length - 1
          ival = ICHAR (dpastr(ii : ii)) - ICHAR ('0')
          IF ((ival.LT.0).OR.(ival.GT.9)) THEN
            CALL Messag ('DPACKD', 3, iretcd, ii, 1, dpastr)
            Dpackd = $8dumy
            GOTO 900
          ELSE
            value = 10.D0 * value + DFLOAT (ival)
          ENDIF
100     CONTINUE
        IF ((dpastr(2 * length : 2 * length).EQ.'A').OR.
     +      (dpastr(2 * length : 2 * length).EQ.'C').OR.
     +      (dpastr(2 * length : 2 * length).EQ.'E').OR.
     +      (dpastr(2 * length : 2 * length).EQ.'F')) THEN
          Dpackd = value
        ELSEIF ((dpastr(2 * length : 2 * length).EQ.'B').OR.
     +      (dpastr(2 * length : 2 * length).EQ.'D')) THEN
          Dpackd = -value
        ELSE
          CALL Messag ('DPACKD', 4, iretcd, ii, 2 * length, dpastr)
          Dpackd = $8dumy
        ENDIF
      ENDIF
900   CONTINUE
      IF (trace) CALL Messag ('DPACKD', 5, 0, 0, 1, Mchrr8 (Dpackd))
      $nest = $nest - 1
      RETURN
    END
```

Vorbereitete definitionen

Folgende INCLUDE bzw COPY members stehen in der bibliothek TWLIB.F77.OBRZ.SOURCE: ... für FORTRAN ... für ASSEMBLER

4.4 Programmskelette

4.4.1 Trace Meldungen

Zur behandlung dieser aufgabe steht in TWLIB.F77.OBRZ.SOURCE das folgende skelett eines programmteiles:

4.4.2 Debug Meldungen

Zur behandlung dieser aufgabe steht in TWLIB.F77.OBRZ.SOURCE das folgende skelett eines programmteiles:

4.4.3 Beispiele von Meldungen

Folgende members in der bibliothek TWLIB.OBRZD.TEXT bzw ..OBRZE.TEXT können zur entwicklung von meldungstexten als muster heran gezogen werden:

4.4.4 Fehlende Parameter

Zur behandlung dieser aufgabe steht in TWLIB.F77.OBRZ.SOURCE das folgende skelett eines programmteiles:

4.4.5 Undefinierte Daten

Zur behandlung dieser aufgabe steht in TWLIB.F77.OBRZ.SOURCE das folgende skelett eines programmteiles:

ABSICHTLICH LEER GELASSEN

A

B

C

D

E

F

G

H

I

K

M

N

P

R

S

T

U

V

ABSICHTLICH LEER GELASSEN