Das DFN Remote Procedure Call Tool

Benutzerhandbuch
Band II - obsolete Kapitel aus Rel. 1.0

Rolf Rabenseifner
Jens Bönisch, Frank Deuble, Chris Dodge, Bernd Hesse, Mandana Karimian
Jan Kayser, Werner Kollak, Hans Dieter Reimann

15. Dezember 1994

Release 1.0.60

Rechenzentrum der Universität Stuttgart
Prof. Dr.-Ing. R. Rühle
Allmandring 30
70550 Stuttgart
(Tel.: 0711-685-5530, FAX 0711-682357)

Gefördert vom
Verein zur Förderung eines Deutschen Forschungsnetzes (DFN),
und Bundesministerium für Forschung und Technologie (BMFT)
unter den Förderkennzeichen TK 558 VA 005.1 und VA 005.3
'Verteilte Anwendungen über Hochgeschwindigkeitsdatenkommunikation'

Interne Dokumentationsbezeichnung

dfnrpc/1.0/src/doc/*.tex (Text)
MAC/rabenseifner/DFNRPC_1.0/pics/* (Fig.-Src)
dfnrpc/1.0/src/doc/pics/*.showcase (Fig.-Src)
dfnrpc/1.0/src/doc/pics/*.eps (EPS-Abb.)

Als Postscript-File zum Ausdrucken:
Printable Postscript Files:

$DFNRPCROOT/dfnrpc/1.0/dfnrpc.ps (Druckvorlage, Band I, deutsch)
$DFNRPCROOT/dfnrpc/1.0/dfnrpc_obsolete.ps (Druckvorlage, Band II, deutsch)
$DFNRPCROOT/dfnrpc/1.0/dfnrpc_e.ps (Druckvorlage, englisch)

Inhalt

COPYRIGHT Notice
The DFN-RPC is a Remote Procedure Call Tool combined with a Message Passing Library.

Copyright ©1994, Rechenzentrum der Universität Stuttgart (RUS) (Computer Center of the University of Stuttgart), Allmandring 30, D-70550 Stuttgart, Germany.
All rights reserved.

All the DFN-RPC software is free software; you can redistribute and/or modify the library libdfnrpc.a and the include files dfnrpc.inc and dfnrpc_stub.inc and the examples in the include and examples directories and their sources under the terms of the GNU Library General Public License and the stub generator dfnrpcgen and dfnrpcm4 and other tools, p.e. the monitor, and their sources under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

These programs and libraries are distributed in the hope that it will be useful, but **without any warranty**; without even the implied warranty of **merchantability** or **fitness for a particular purpose**. See the GNU Library General Public License and GNU General Public License for more details.

You should have received a copy of the GNU General Public License and the GNU Library General Public License along with this program, it should be in the files named COPYING-2.0 and COPYING.LIB-2.0; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA, or fetch them from ftp://prep.ai.mit.edu/pub/gnu.

Among other things, the copyright notice and this notice must be preserved on all copies.

Projektleitung: Rolf Rabenseifner. Projektstart: 1. Sept. 1988

Gefördert vom
Verein zur Förderung eines Deutschen Forschungsnetzes (DFN) und Bundesministerium für Forschung und Technologie (BMFT) unter dem Förderkennzeichen TK 558 VA 005 „Verteilte Anwendungen über Hochgeschwindigkeitsdatenkommunikation“ und TK 558 VA 005.3 „Erweitertes RPC-Tool für verteilte Anwendungen“.

14 Object Handling von Rel. 1.0 - obsolet

14.1 Übersicht

Mit dem Object Handling werden die Blöcke auf entfernten Rechnern gestartet und die entsprechenden Kommunikationsverbindungen aufgebaut, damit der DFN-RPC die nötigen Nachrichten für die Remote Procedure Calls austauschen kann.

Das Object Handling muß von der Anwendung durch Bibliotheksfunktionen aufgerufen werden. Diese Aufrufe werden *nicht* vom Stub-Generator generiert. Vielmehr generiert er für jeden Block einen Satz dieser Subroutinen, da sie auch anwendungsspezifische Deklarationen enthalten, wie z.B. die Länge von Send- und Receive-Buffer.

Es empfiehlt sich, die Aufrufe der Object Handling Routinen in speziellen Object Client-Procedure- und Server-Program-Modules (cp und sp in den Beispielen) zusammenzufassen.

Einführende Beispiele findet man im User's Guide Kapitel 4.8 auf Seite 42.

Ein Block besteht aus allen Modulen, die bei Laufzeit zum selben Prozeß gehören. Blöcke müssen mit RPBINI initialisiert und mit RPBFIN beendet werden.

Nach der Initialisierung muß ein Transport Service mit RPSINI initialisiert und vor RPBFIN mit RPSFIN wieder geschlossen werden. Falls mehrere Server aufgebaut werden sollen, *kann* man RPSINI auch öfters aufrufen. Wenn man hierzu verschiedene Transport Services benutzen will, dann *muß* man RPSINI öfters aufrufen.

Nachdem der Transport Service eröffnet ist, kann der Master seinen Slave mittels der Object-Start-Routinen RPO... starten.

Im folgenden werden die Begriffspaare Client/Server und Master/Slave folgendermaßen eingesetzt:

Client — Der Client enthält das Hauptprogramm der Applikation. Diese ruft mittels RPC Routinen im Server auf. Er enthält typischerweise zwei Unterprogramme, rpcini und rpcfin, welche vom Anwender zu schreiben sind und die Object Handling Routinen aufrufen.

Server — Der Server enthält den Block Dispatcher RPBD1, welcher wartet, bis RPCs ankommen und dann veranlaßt, daß die entsprechende Routine aufgerufen wird.

Master — Der Master sorgt dafür, daß der Slave gestartet wird. Hierzu benutzt er die RPO..I und RPO..S Routinen, denen er einen String übergibt, welcher mittels eines allgemeinen Betriebssystemaufrufs beliebige Dienstprogramme für den Slave-Start einsetzen kann. Die RPO..S Routine wartet, bis der Slave gestartet ist. Der Master liegt typischerweise auf der Workstation.

Slave — Der Slave wird vom Master gestartet. Seine RPO..S Routine synchronisiert sich mit der des Masters.

Beim Objektstart gibt es folgende Fälle und entsprechende Programmierschnittstellen:

```
CALL RPO{Case}{Site}{Phase}
          1     C     I
          2     S     S
          3           F

Case:  1 = Client=Master, Server=Slave,  without restart
```

```
       2 = Client=Master, Server=Slave,  with restart
       3 = Client=Slave,  Server=Master
Site:  C = Client
       S = Server
Phase: I = Initialize the start (only with Case=2)
       S = Start the slave and establish the connection
       F = Finish the object, i.e. close the connection

Case | Master (Works.)|                 | Slave (Superc.)
---------------------------------------------------------
  1  | Client  RPO1CS | ---> start ---> | Server  RPO1SS
     |                |      +port_info |
     |                |      +auth_info |
     |                | type_i_auth->rd |
     |                | <-- connect <-- |
     |                | i_auth_tst | -- |
     |                |                 |
     |         RPCalls| -->->->->->->-> |         RPBD1
     |                |                 |
     |         RPO1CF | --> disconn --> |         RPO1SF
---------------------------------------------------------
  2  | Client  RPO2CI | ---> start ---> | Server  RPO2SI
     |                |      +port_info |
     |                | <-- connect <-- |
     |                | -- | open port_2|
     |                | <- port_num_2 <-|
     |                | store p_num_2|--|
     |                | --> disconn --> |
     |         RPO2CS | read p_num_2 |--|         RPO2SS
     |                | --> connect --> |
     |                |                 |
     |         RPCalls| -->->->->->->-> |         RPBD1
     |                |                 |
     |         RPO2CF | ->restart_flg-> |         RPO2SF
     |                | --> disconn --> |
     |                | --|close port_2 | (if no restart)
---------------------------------------------------------
  3  | Server  RPO3SS | ---> start ---> | Client  RPO3CS
     |                |      +port_info |
     |                |      +auth_info |
     |                | type_i_auth->rd |
     |                | <-- connect <-- |
     |                | i_auth_tst | -- |
     |                |                 |
     |         RPBD1  | <-<-<-<-<-<-<-- |         RPCalls
     |                |                 |
     |         RPO3SF | <-- disconn <-- |         RPO3CF
---------------------------------------------------------
```

Außerdem gibt es noch folgende Utilities:

```
RPSLTI  -  list all transport interfaces
RPSUSE  -  switch to an alternate and already opened service
RPOFLS  -  flush the buffer after buffered asynchronous calls
```

```
RPOUSE  -  change current connection to an alternate obj.
RPOSYN  -  an empty synchronization call
RPOGSR  -  get send and receive buffer length
RPOSSR  -  set send and receive buffer length
```

Zusammenfassung aller Object Handling Routinen:

```
SUBROUTINE RPBINI  - block init
SUBROUTINE RPBFIN  - block finish
SUBROUTINE RPBD1   - block dispatcher
SUBROUTINE RPSLTI  - list all transport interfaces
SUBROUTINE RPSINI  - init of transp. service
SUBROUTINE RPSFIN  - fin. of transp. service
SUBROUTINE RPSUSE  - switch to an alternate service
SUBROUTINE RPOFLS  - flush  one object
SUBROUTINE RPOUSE  - flush a. switch to altern.object
SUBROUTINE RPOSYN  - an empty synchronization call
SUBROUTINE RPOGSR  - get send and receive buffer length
SUBROUTINE RPOSSR  - set send and receive buffer length
SUBROUTINE RPO1CS  - object client master norm. start
SUBROUTINE RPO1SS  - object server slave  norm. start
SUBROUTINE RPO1CF  - object client master normal fin.
SUBROUTINE RPO1SF  - object server slave  normal fin.
SUBROUTINE RPO2CI  - object client master initiali.
SUBROUTINE RPO2SI  - object server slave  initiali.
SUBROUTINE RPO2CS  - object client master (re)start
SUBROUTINE RPO2SS  - object server slave  (re)start
SUBROUTINE RPO2CF  - object client mst. fin./restart
SUBROUTINE RPO2SF  - object server slv. fin./restart
SUBROUTINE RPO3SI  - object server master initiali.
SUBROUTINE RPO3SS  - object server master (re)start
SUBROUTINE RPO3CS  - object client slave  (re)start
SUBROUTINE RPO3SF  - object server mst. fin./restart
SUBROUTINE RPO3CF  - object client slv. fin./restart
```

14.2 Einführungsbeispiel ua1a

Der DFN-RPC ist ein *Remote Procedure Call*-Tool für in Fortran oder C geschriebene Programme.

Um den Einstieg in die Verwendung des DFN-RPC zu erleichtern, wird ein einfaches Beispiel mit den grundlegenden Elementen einer verteilten Anwendung beschrieben.

Dabei wird insbesondere der Weg von der unverteilten, lokalen Anwendung hin zum verteilten Programm Schritt für Schritt aufgezeigt.

Die Aufgabe unseres Einführungsbeispiels ist die Ausgabe eines Datensatzes auf den Bildschirm und einen File eines anderen Rechners.

Wir nennen dieses Beispiel *ua1a* und stellen als erstes die unverteilte, also lokale Version ohne DFN-RPC vor. Die verwendete Programmiersprache ist Fortran.

Anmerkung: Deklarationen und Initialisierungen der verwendeten Variablen werden nicht explizit dargestellt. Der folgende Code ist auf das momentan Wesentliche reduziert und daher nicht direkt lauffähig!

```
program ua1a
  open  (lun,file='ua1.dat',status='unknown')
  write (lun,'...') data
  write (*,  '...') data
  close (lun)
  stop
end
```

Voraussetzung für das Verteilen einer Anwendung ist ein *rein prozedurales Interface*, da ja nur Subroutine-Calls und deren Parameter über das Netz geschickt werden. Daher muß das Programm gegebenenfalls so modifiziert werden, daß

- alle Aufgaben, die auf dem anderen Rechner ausgeführt werden sollen, in Subroutinen und Functions verpackt sind (*prozedurales Interface*) und
- alle Daten zwischen Hauptprogramm und aufgerufenen Routinen nur über die Parameterliste im Subroutine-Call ausgetauscht werden (*rein* prozedural). Es darf also keine Common-Blöcke mit globalen Variablen geben, die von beiden Programmteilen aus benützt werden!

Bei komplexeren Anwendungen stellt sich bald die Frage, wie man diese am sinnvollsten verteilt. Meist berücksichtigt man dabei die Geschwindigkeit der verwendeten Rechner und Netzwerke.

Aber auch die schon existierende modulare Struktur der Software spielt bei der Verteilung bestehender Anwendungen eine große Rolle.

Werden diese Überlegungen auf unser Beispiel angewendet, ergibt sich folgendes Programmsegment:

```
program ua1a
  call ropen1  (lun,'ua1.dat','unknown')
  call rwrite1 (lun,'...',data)
  call rclose1 (lun)
  stop
end

subroutine ropen1 (lun,name,stat)
  open  (lun,file=name,status=stat)
  return
end

subroutine rwrite1 (lun,fmt,data)
  write (lun,fmt) data
  write (*,  fmt) data
  return
end

subroutine rclose1 (lun)
  close (lun)
  return
end
```

Hier wird die Aufteilung in die zwei logischen Module mit dem Hauptprogramm einerseits und den Subroutinen ropen1, rwrite1 und rclose1 andererseits schon deutlich sichtbar. Das Hauptprogramm läeuft späeter in der Regel auf der Workstation, der sogenannten *Client_Site*. Die Subroutinen werden auf den meist leistungsstarken Compute-Server, die *Server_Site*. ausgelagert.

Anmerkung: Client_Site ist immer dort, wo sich das Hauptprogramm der Anwendung befindet.

Für die Kommunikation per Netzwerk und das Aufrufen von ausgelagerten Routinen in einem anderen Prozeß, der meist auch auf einem anderen Rechner läuft, sind zusätzliche Funktionalitäten erforderlich. So werden neben den sogenannten *Stubs*, die die Subroutine-Calls über das Netz weiterleiten, noch Initialisierungs- und Abschlußroutinen für die Transport- und DFN-RPC-Dienste benötigt.

Das Server-Modul erhält ein spezielles Hauptprogramm, das die eintreffenden Aufrufe an die entsprechenden Subroutinen weiterleitet. Dieses Hauptprogramm wird auch *Dispatcher* genannt.

Damit diese Funktionalitäten zum überwiegenden Teil von DFN-RPC-Tools erzeugt werden, müssen zusätzlich beschreibende Statements bereitgestellt werden, die vom sogenannten *Stub-Generator* abgearbeitet werden.

Single Source Prinzip, Blöcke und Module:

In diesem Einführungsbeispiel werden alle für den RPC und die Verteilung des Anwendungsprogramms nötigen Informationen, Änderungen, etc. in den ursprünglichen Source-File integriert (*Single Source Prinzip*). Man kann diese Information auch getrennt bereitstellen, siehe hierzu Kapitel 5.5 auf Seite 84.

Ein *Modul* ist eine Menge von Subroutinen und Funktionen, die aus der Sicht des Anwendungsprogrammierers eine logische Einheit bilden.

Der Stubgenerator hat zwei Aufgaben,

- das *Sourcecode Handling*, d.h. das Zusammenstellen der Fortran-Sourcen der Anwendung im unverteilten Fall (komplette Anwendung ohne RPC-Besonderheiten) für einen Rechner (sog. *Local_Site*) und im verteilten Fall (aufgeteilte Software mit besonderen Anweisungen für den DFN-RPC) für mehrere Rechner (sog. *Client_Site* und *Server_Sites*)
- das *Generieren* der sog. *Stubs* (deutsch: Stummel), die die Remote Procedure Calls über die Leitung schicken.

Hierzu müssen für die Anwendung *Blöcke* definiert werden. Ein *Block* besteht aus mehreren Modulen und wird zu einer Einheit zusammengebunden (z.B. unter Unix mit dem ld). Der Inputfile für den Stubgenerator (hier *ua1a.rpi*) enthält hierzu *Block Definitions*. Diese wiederum enthalten Verweise auf die zugehörigen Module.

Wenn man seine Anwendung unverteilt laufen lassen will, muß man einen sogenannten Local-Site Block erstellen, welcher alle Anwendungsmodule enthält (wobei diese das Hauptprogramm und alle Unterprogramme enthalten).

```
C*[MOD_BEG] cm, NAME='ua1 client module - main program'       (module begin directive)
      program rpitst
        call ropen1  (lun,'ua1.dat','unknown')
        call rwrite1 (lun,'...',data)
        call rclose1 (lun)
        stop
      end
C*[MOD_END]                                                   (module end directive)

C*[MOD_BEG] sm, NAME='ua1 server routines'                    (module begin directive)
      subroutine ropen1 (lun,name,stat)
        ...
      subroutine rwrite1 (lun,fmt,data)
        ...
      subroutine rclose1 (lun)
        ...
C*[MOD_END]                                                   (module end directive)
```

Der Block der unverteilten Anwendung wurde *lb* genannt. Hierfür ist im Inputfile ua1a.rpi folgende Block Definition voranzustellen (s.a. Abb. 14.1 auf S. 212):

```
C*[BLK_BEG] lb, NAME='ua1 local block', LOCAL_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     sm, LOCAL
C*[BLK_END]
```

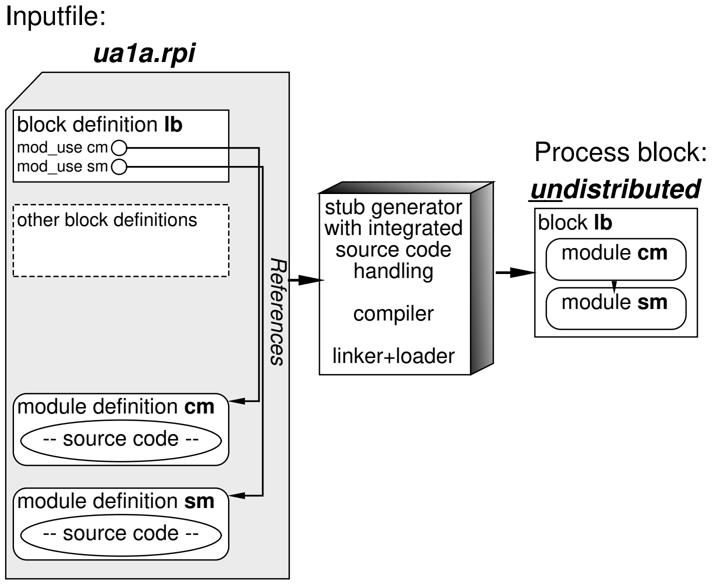

Abbildung 14.1: Der Local-Site-Block des Beispiels ua1a

Nun wird das Programm verteilt:

Soll die Anwendung verteilt laufen, muß man einen Client-Site Block, der das Modul mit dem Hauptprogramm der Anwendung enthält, sowie einen Server-Site Block erstellen, der die Module mit jenen Unterprogrammen enthält, die ausgelagert gerechnet werden sollen, siehe auch Abb. 14.2 auf Seite 213.

Gibt man den Modulen die Namen *cm* und *sm*, dann enthält der Inputfile für den Stubgenerator (hier *ua1a.rpi*) folgende *Module Definitions*:

```
C*[BLK_BEG] cb, NAME='ua1 client block', CLIENT_SITE
C*[MOD_USE]     cm, LOCAL
...
C*[MOD_USE]     sm, CLIENT                  (hierdurch wird nur der Client-Stub generiert,
C*[BLK_END]                              die Anwendungsroutinen sind im Server-Block sb)

C*[BLK_BEG] sb, NAME='ua1 server block', SERVER_SITE
...
C*[MOD_USE]     sm, SERVER                        (Server-Stub + Anwendungsroutinen)
C*[BLK_END]
```

Das Hauptprogramm soll nun auf dem einen Rechner (*Client_Site*) laufen und die Subroutinen ropen1, rwrite1 und rclose1 auf dem anderen (*Server_Site*). Der DFN-RPC hat dafür zu sorgen, daß die Calls von der Client_Site über das Netz zur Server_Site gehen.

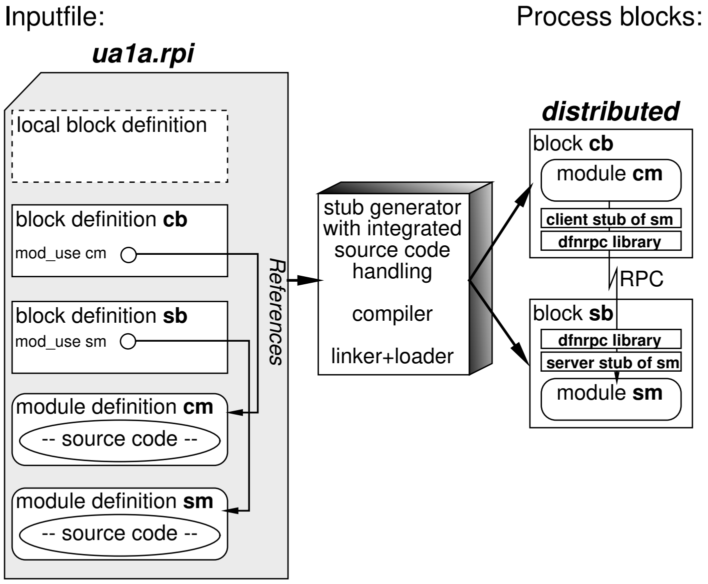

Abbildung 14.2: Client- und Server-Site-Block

Im Detail sind beim Verteilen eines Programmes folgende Schritte notwendig:

1. Zusammenstellen der Blöcke,
2. Hinzufügen von speziellen Init- und Finish-Aufrufen und Subroutinen im Client_Site Block,
3. Hinzufügen eines speziellen Hauptprogramms zum Server_Site Block,
4. Hinzufügen der *Remote Procedure Interface Definition* (**RPI**) zum Server-Modul.
5. Aufruf des Stubgenerators
6. Kompilieren und Binden

Abb. 14.3 auf Seite 214 stellt die im folgenden genauer beschriebenen Schritte detailliert dar.

1. Schritt: <u>Zusammenstellen der Blöcke</u>

   Für die Client_Site und die Server_Site muß jeweils ein Block erstellt, kompiliert und gebunden werden. Der Client_Site Block (hier mit *cb* benannt) enthält das Modul cm, der Server_Site Block (*sb*) das Modul sm:

```
C*[BLK_BEG] cb, NAME='ua1 client block', CLIENT_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     cp, LOCAL                                  (vgl. 2. Schritt)
C*[MOD_USE]     sm, CLIENT          (hierdurch wird nur der Client-Stub generiert,
C*[BLK_END]                       die Anwendungsroutinen sind im Server-Block sb)

C*[BLK_BEG] sb, NAME='ua1 server block', SERVER_SITE
C*[MOD_USE]     sp, LOCAL                                   (vgl 3. Schritt)
C*[MOD_USE]     sm, SERVER                 (Server-Stub + Anwendungsroutinen)
C*[BLK_END]
```

   Mit dem Statement C*[BLK_BEG] cb, NAME='ua1 client block', CLIENT_SITE schreibt

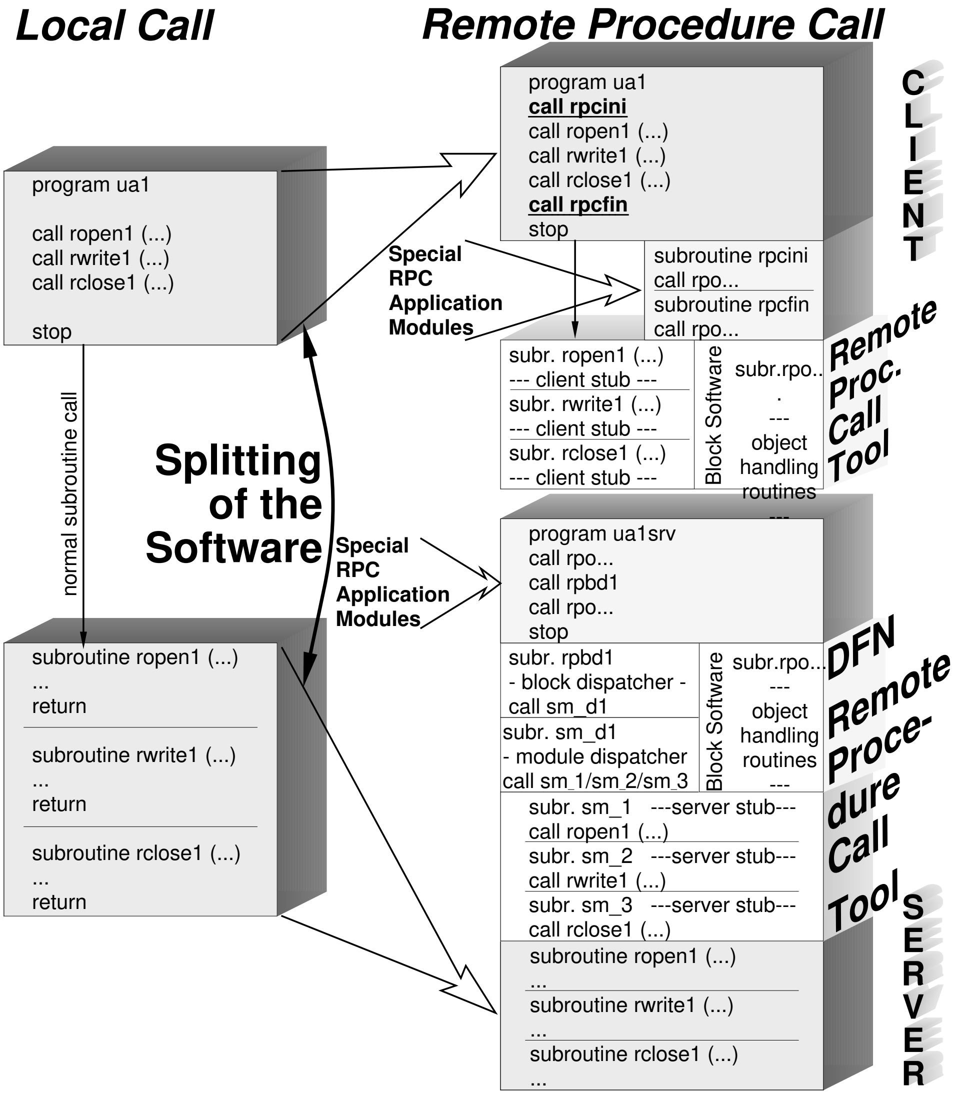

Abbildung 14.3: Beispiel ua1a vom 'local call' zum 'remote procedure call'

der Stubgenerator die client-seitige allgemeine DFN-RPC Block-Software auf den Stub-Presourcefile[1] ua1a_cb_stb.m4.

Durch das Statement C*[MOD_USE] cm, LOCAL wird im Client_Site Block cb das Modul cm eingefügt, d.h. daß die Fortransource dieses Moduls auf den File ua1a_cb.f geschrieben wird.

Außerdem sind die Client-Stubs für das Server-Modul sm zu generieren und einzufügen. Das Statement C*[MOD_USE] sm, CLIENT bewirkt zum einen, daß das Modul sm in der Block-Software bekannt ist (für Fehlermeldungen wichtig) und zum anderen, daß die Client-Stubs zum Modul sm generiert und ebenfalls auf die Files ua1a_cb_stb.m4, bzw. ua1a_cb_stb.f geschrieben werden. (Siehe auch Abb. 14.3 auf Seite 214.)

[1] Die Stubs werden zweistufig generiert: auf der Basis der in den Schritten 1-4 zu erstellenden Remote Interface Definition werden in einem ersten Schritt die sog. Stub-Presourcefiles erzeugt. Diese enthalten kompakte Makro-Aufrufe. In der zweiten Stufe wird dieser ebenfalls vom Stubgenerator in den Fortranfile ua1a_cb_stb.f umgesetzt.

Mit C*[BLK_BEG] sb, NAME='ua1 server block', SERVER_SITE wird die server-seitige Block-Software auf die Files ua1a_sb_stb.m4, bzw. ua1a_sb_stb.f geschrieben.

Durch das Statement C*[MOD_USE] sm, SERVER wird die im Server_Site Block sb das Modul sm eingefügt, d.h. daß die Fortransource dieses Moduls auf den File ua1a_sb.f geschrieben wird.

Zusätzlich wird das Modul sm in der Block-Software eingetragen (z.B. im Block-Dispatcher[2]), die Server-Stubs zum Modul sm generiert und diese auf die Files ua1a_cb_stb.m4 und ua1a_cb_stb.f geschrieben.

2. Schritt: Spezielle Init- und Finish-Aufrufe und Subroutinen im Client_Site Block

   Will man eine Anwendung verteilt auf zwei Rechner, d.h. genauer in zwei Prozessen laufen lassen, dann muß man dafür sorgen, daß *beide gestartet* werden und daß eine Netzwerkverbindung zwischen ihnen aufgebaut wird. Der DFN-RPC stellt hierfür die *Object Handling* Subroutinen zur Verfügung.

   Im Beispiel ua1a sieht das client-seitig so aus:

```
C*[MOD_BEG] cm, NAME='ua1 client module - main program'
      program rpitst

C*RPC   call rpcini

        call ropen1  ...
        call rwrite1 ...
        call rclose1 ...

C*RPC   call rpcfin

        stop
      end
C*[MOD_END]
```

   Zeilen, die mit C*RPC beginnen, werden in Client_Site und Server_Site, nicht aber Local_Site Blöcken in richtige Fortranstatements umgewandelt.

   Die beiden Subroutinen *rpcini* und *rpcfin* sind Teil der Anwendung und bilden normalerweise ein weiteres Modul - hier mit *cp* benannt (gekürzte Darstellung):

```
C*[MOD_BEG] cp, NAME='ua1 client rpc special routines'
      SUBROUTINE RPCINI
       ...
       RETURN
      END

      SUBROUTINE RPCFIN
       ...
       RETURN
      END
C*[MOD_END]
```

   Die Details sind im Kapitel 4.8 auf Seite 42 beschrieben. Für die erste eigene verteilte Anwendung empfiehlt es sich, dieses Modul aus dem mitgelieferten Beispiel ua1a (oder ben) herauszukopieren.

[2] Dieser verzweigt einen über das Netzwerk ankommenden Remote Procedure Call zum entsprechenden Modul-Dispatcher, welcher weiter zum Stub der entsprechenden Subroutine verzweigt, welcher dann diese Subroutine ruft.

Mit dem Statement C∗[MOD_USE] cp, LOCAL wird es in den Client_Site Block cb eingefügt (s.o.).

Alternativ zur bedingten Kompilation der Statements `call rpcini` und `call rpcfin` mittels der DFN-RPC-Direktive C∗RPC kann man auch im unverteilten Fall zwei Dummy-Routinen rpcini und rpcfin hinzufügen. Diese würde man in ein weiteres Modul stellen, welches im Block *lb* mittels MOD_USE ebenfalls eingebunden werden muß.

3. Schritt: Spezielles Hauptprogramm zum Server_Site Block

Ein spezielles Hauptprogramm stellt server-seitig das Gegenstück zu den client-seitigen Routinen rpcini und rpcfin dar. Es ist Teil der Anwendung und bildet normalerweise ein weiteres Modul - hier mit *sp* benannt (gekürzte Darstellung):

```
C*[MOD_BEG] sp, NAME='ua1 server rpc special program'
      PROGRAM UA1SRV
       ...
       STOP
      END
C*[MOD_END]
```

Die Details sind im Kapitel 4.8 auf Seite 42. beschrieben. Für die erste eigene verteilte Anwendung empfiehlt es sich, dieses Modul ebenfalls aus dem mitgelieferten Beispiel ua1a (oder ben) herauszukopieren.

Mit dem Statement C∗[MOD_USE] sp, LOCAL wird es in den Server_Site Block sb eingefügt (s.o.)

4. Schritt: Remote Procedure Interface Definition zum Server-Modul

Dem DFN-RPC Stubgenerator muß man mitteilen, welche Subroutine-Calls er als RPC durchführen soll, d.h. zu welchen Subroutinen er Client- und Server-Stubs generieren soll[3].

Beim DFN-RPC kann diese Information, die sog. *Remote Procedure Interface Definition*, in die Subroutine-Header integriert werden. Die Call-Statements bleiben unberührt.

In unserem Beispiel steht dann (etwas vereinfacht):

```
C*[MOD_BEG] sm, NAME='ua1 server routines'
C*[RPI_BEG] ASYNCHRONOUS
      subroutine ropen1 (lun,filestring,statusstring)
        character*(*) filestring   [IN]
        character*(*) statusstring [IN]
        integer lun                [IN]
C*[RPI_NON]
        open(lun,file=filestring,status=statusstring)
        return
      end
C*[RPI_END]

C*[RPI_BEG]
      subroutine rwrite1 (lun,formatstr,iwrite,rwrite,satzlaenge)
        character*(*) formatstr    [IN]
```

[3] Client-Stubs werden auf der Seite benötigt, wo der Call stattfindet, Server-Stubs dort, wo die Anwendungs-Subroutine ist.

```
        integer satzlaenge         [IN]
        integer lun                [IN]
        integer iwrite(satzlaenge) [IN]
        real    rwrite(satzlaenge) [IN]
C*[RPI_NON]
        write(lun,formatstr) (iwrite(i),rwrite(i),i=1,satzlaenge)
        return
      end
C*[RPI_END]

C*[RPI_BEG]
      subroutine rclose1 (lun)
        integer lun [IN]
C*[RPI_NON]
        close(lun)
        return
      end
C*[RPI_END]
C*[MOD_END]
```

Es gibt zwei Möglichkeiten, Variablen zu deklarieren :

```
      character*(*) filestring   [IN]        (entweder inline)

      character*(*) filestring
C*RPC+                           [IN]        (oder als Kommentar-Direktive)
```

Die Details hierzu sind im Kapitel 5.3.6 auf Seite 69 beschrieben.

5. Schritt: Aufruf des Stubgenerators

   Zuerst muß man den in den Schritten 1-4 angegebenen Code auf einen File schreiben, den RPI-File. Normalerweise sollte er die Endung *.rpi* haben, in unserem Beispiel heißt er *ua1a.rpi*.
   Wenn man will, kann man zusätzlich zur Information zur Generierung des Client_Site Blocks und des Server_Site Blocks (cb,sb) noch jene zur Erzeugung des Local_Site Blocks (lb) einfügen. Der File *ua1a.rpi* ist vollständig auf Seite 227 abgedruckt.
   Der Stubgenerator erstellt die Fortranfiles für die einzelnen Blöcke, siehe Abb. 14.4 auf Seite 219.

Mit dem Kommando

$DFNRPCBIN/dfnrpcgen ua1a

werden auf der Basis des Inputfiles *ua1a.rpi* folgende Files erzeugt:

Block	File	Filetype	Inhalt	
lb	ua1a_lb.f	Application-Source	Module	cm + sm
cb	ua1a_cb.f	Application-Source	Module	cm + cp
	ua1a_cb_stb.m4	Stub-Presource	Block-Software	zu cb
			Client-Stub	zu sm
	ua1a_cb_stb.f	Stub-Source	Block-Software	zu cb
			Client-Stub	zu sm
sb	ua1a_sb.f	Application-Source	Module	sp + sm
	ua1a_sb_stb.m4	Stub-Presource	Block-Software	zu sb
			Server-Stub	zu sm
	ua1a_sb_stb.f	Stub-Source	Block-Software	zu sb
			Server-Stub	zu sm

Die Stub-Presourcefiles[4] sind Scratchfiles und für die weiteren Schritte ohne Bedeutung.

6. Schritt: Kompilieren und Binden

Alle Fortran Files *xxx.f* eines Blocks sind ganz normal zu kompilieren und zusammenzubinden. Zur Kompilation der Subroutinen RPCINI und RPCFIN benötigt man den Includefile dfnrpc.inc und zur Kompilation der Stubs den Includefile dfnrpc_stub.inc. Diese kann man sich zugänglich machen mit

ln -s $DFNRPCINCLUDE/dfnrpc.inc dfnrpc.inc

ln -s $DFNRPCINCLUDE/dfnrpc_stub.inc dfnrpc_stub.inc

Bei den Blöcken der verteilten Anwendung ist grundsätzlich die DFN-RPC Bibliothek (*Library*) hinzuzuladen:

$(DFNRPCLIB)/libdfnrpc.a

Außerdem ist/sind grundsätzlich die Bibliothek(en) für die Transportsoftware hinzuzuladen. Beim TCP/IP Protokollstack, welcher grundsätzlich benötigt wird, geschieht dies automatisch, außer auf folgenden Rechnern:

Protokollstack	Manufacturer	loader option
TCP/IP	cray	-lnet
	sg (silicon graphics)	-lbsd

Bei zusätzlichen Protokollstacks sind hinzuzuladen:

Protokollstack	Manufacturer	loader option
ULTRA = true	convex	-lulsock
	cray	-lulsock
	sun	/usr/lib/libulsock.a
OSI = cray	cray	-ltli
OSI = sun	sun	(nichts)

Bei den mitgelieferten und für den Test des DFN-RPC entwickelten Beispielen werden folgende Aufrufe und Optionen benützt. Sie gelten vorallem auch für große Bibliotheken, wie z.B. das mit dem DFN-RPC verteilte PHIGS.

[4] siehe Seite 214

manu-facturer	archi-tecture	operating system	fortran compiler	compile options	loader	load options	net lib
alliant apollo	FX DN10000	Concentrix 5.0 SR 10.2	f77 /com/ftn	-Nx2000 -72 -indexl	f77 ld		
convex cray	C120 cray2	ConvexOS unicos 4.0, 5.0, 6.0	f77 cf77	-72 -mf "-a static"	f77 segldr		 -lnet
dec	dec vax vax	ultrix ultrix vms	f77 fort fort	-Nn2000 -Nl1000	f77 fort link		
ibm sg	RS6000 iris3d iris4d	AIX IRIX IRIX 3.2	xlf f77 f77	-qcharlen = 8192 -Zf -Nn2000 -Nl1000 -Nn2000 -Nl1000	xlf f77 f77	 -lc -ldbm	 -lbsd -lbsd
sun sun	sun3 sun4	SunOS 3.5, 4.0.3, 4.1 SunOS 3.5, 4.0.3, 4.1	f77 f77	-Nx2000 -Nl1000 -Nx2000 -Nl1000	f77 f77		

Bei dec/dec/ultrix muß f77 ein Link auf den Compiler „fort“ sein.

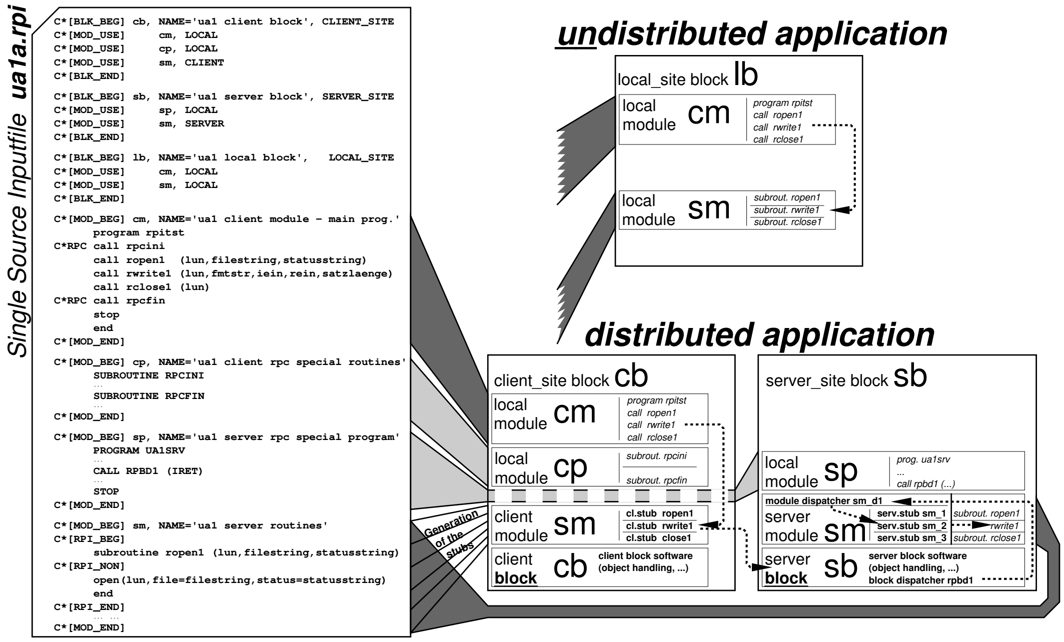

Abbildung 14.4: Vom rpi-File zum Anwendungs-Block

14.3 Danach die Praxis

Um den DFN-RPC einsetzen zu können, müssen folgende *environment Variablen* gesetzt sein[5] - Beispiel:

Environment Variable	Directory
$DFNRPCROOT	root directory zum DFN-RPC z.B. /usr/3rdparty/dfn
$DFNRPCCONF[6]	configuration directory z.B. $DFNRPCROOT/dfnrpc/1.0/sys/sun4 oder $DFNRPCROOT/dfnrpc/1.0/sys_debug/sun4
$DFNRPCBIN	bin directory des DFN-RPC i.A. $DFNRPCCONF/bin
$DFNRPCLIB	lib directory des DFN-RPC i.A. $DFNRPCCONF/lib
$DFNRPCINCLUDE	include directory des DFN-RPC i.A. $DFNRPCCONF/include
$DFNRPCEXAMPLES	examples directory des DFN-RPC i.A. $DFNRPCCONF/examples_ieee
$DFNRPCSRC[7]	source directory des DFN-RPC i.A. $DFNRPCROOT/dfnrpc/1.0/src

Die Werte von DFNRPCROOT und DFNRPCCONF erfahren Sie von Ihrem Systemadministrator oder demjenigen, der den DFN-RPC installiert hat. Falls er noch nicht installiert ist, dann sei auf Kapitel 2 auf Seite 3 verwiesen.

Die Environment Variablen kann man in einer C-Shell setzen mit (Beispiel):

```
setenv DFNRPCROOT     /usr/3rdparty/dfn
setenv DFNRPCSRC      $DFNRPCROOT/dfnrpc/1.0/src
setenv DFNRPCCONF     $DFNRPCROOT/dfnrpc/1.0/sys/sun4
setenv DFNRPCBIN      $DFNRPCCONF/bin
setenv DFNRPCLIB      $DFNRPCCONF/lib
setenv DFNRPCINCLUDE  $DFNRPCCONF/include
setenv DFNRPCEXAMPLES $DFNRPCCONF/examples_ieee
```

Falls in $DFNRPCEXAMPLES/* nur die makefiles vorhanden sind, so wurden die Beispiele bei der Installation nicht kompiliert.

14.3.1 Das erste Experiment mit ua1a

Machen Sie zwei Fenster mit einer C-Shell auf, sorgen Sie dafür, daß die Environmentvariablen gesetzt sind und los geht's.

[5] Alle Beispiele sind in einer C-Shell (/bin/csh) beschrieben - man kann sie aber auch in einer Bourne-Shell (/bin/sh) ausführen.

[6] Sie dient nur als Abkürzung in den nachfolgenden Deklarationen

[7] Sie steht dem Anwender nicht zur Verfügung

Experiment 1:

Linkes Fenster	Rechtes Fenster
% cd $DFNRPCEXAMPLES/ua1a	*% cd $DFNRPCEXAMPLES/ua1a*
% ua1a_cb	
rpcini - please type Local Hostname	
localhost (oder TCP/IP-Namen des lokalen Hosts = linkes Fenster)	
rpcini - please type Remote Hostname	
localhost (oder TCP/IP-Namen des entfernten Hosts = rechtes Fenster)	
rpo1cs - Information for the Slave: **Start_Flags = 00008224** **Master_Host = localhost** **Master_Tsap = 3644**	
rpo1cs - Object_Start_String = **Please start ua1*_sb** **on localhost**	
rpo1cs - Please START the slave **process characterized by** **the Object_Start_String**	
(Client ua1a_ wartet, bis Slave ua1a_sb im rechten Fenster initialisiert ist)	
	% **ua1a_sb** (wie im linken Fenster im Object_Start_String verlangt)
↓ ↘ →→	**rpo1ss - Please type the Start_Flags?** *8224* (wie im linken Fenster oben angezeigt)
↓ ↘ →→	**rpo1ss - Please type the Master_Host?** *localhost* (wie im linken Fenster oben angezeigt)
↓ ↘ →→	**rpo1ss - Please type the Master_Tsap?** *3644* (wie im linken Fenster oben angezeigt)
rpcini - rpc correctly initialized	**ua1srv - rpc correctly initialized**
	ua1srv - starting rpc main dispatcher **open 10 ua1.dat unknown** **write 10(4(i5,f5.1)) 1 1.0...** **write 10(4(i5,f5.1)) 5 5.0...** ... ... **write 10(4(i5,f5.1)) 77 77.0...** **close 10**
	ua1srv - rpc main dispatcher correctly executed
rpcfin - rpc correctly finished	**ua1srv - rpc correctly finished**
%	%

Wenn dieses Experiment erfolgreich war, dann sollte man es mit einer selbst kompilierten Fassung probieren.

Falls Probleme auftreten, so sollte man zuerst die unverteilte Version starten:

```
% cd $DFNRPCEXAMPLES/ua1a
% ua1a_lb
```

14.3.2 Beispiel ua1a selbst kompiliert

Sie eröffnen in Ihrer Homedirectory eine Subdirectory dup_ua1a. In dieser wird nun das ua1a nochmals kompiliert.

Experiment 2:

```
% cd
% mkdir dup_ua1a
% cd    dup_ua1a
% cp $DFNRPCEXAMPLES/ua1a/makefile .
% cp $DFNRPCEXAMPLES/ua1a/ua1a.rpi .
% cp $DFNRPCEXAMPLES/ua1a/ua1a_cb_stb.m4 .
% cp $DFNRPCEXAMPLES/ua1a/ua1a_sb_stb.m4 .
% make

/usr/3rdparty/dfn/dfnrpc/1.0/sys/sun4/bin/dfnrpcgen  -T ieee ua1a
dfnrpcgen option type = -T
          option      = ieee                (Aufruf des DFN-RPC Stubgenerators)
          application = ua1a                (er braucht mindestens 60 Sekunden)
 make: compilation   ua1a_lb.f--->ua1a_lb.o
ua1a_lb.f:
 MAIN rpitst:
        ropen1:                             (Der local_site Block lb wird
        rwrite1:                             kompiliert und gebunden)
        rclose1:
make module         ua1a_lb.o--->ua1a_lb

rm -f dfnrpc.inc                    (dfnrpc.inc wird zugaenglich)
ln -s /usr/3rdparty/dfn/dfnrpc/1.0/sys/sun4/include/dfnrpc.inc dfnrpc.inc

make: compilation   ua1a_cb.f--->ua1a_cb.o
ua1a_cb.f:                          (Client_site Block cb ...)
 MAIN rpitst:                       (...Application Module cm)
        rpcini:                             (...Application Module cp)
        rpcfin:
make: compilation   ua1a_cb_stb.f--->ua1a_cb_stb.o
ua1a_cb_stb.f:
        rphiod:                             (...Block Software zum cb)
        rpbini:
        rpbfin:
        rpsini:
        rpsfin:
        rpo1cs:
        rpo1cf:
        .......                             (.....49 weitere Routinen)
        ropen1:                             (...Client Stub Module sm)
        rwrite1:
        rclose1:
        sm_nam:
```

```
        sm_ini:
make module      ua1a_cb.o--->ua1a_cb

make: compilation   ua1a_sb.f--->ua1a_sb.o
ua1a_sb.f:                            (Server_site Block sb ...)
 MAIN ua1srv:                         (...Application Module sp)
        ropen1:
        rwrite1:                              (...Application Module sm)
        rclose1:
make: compilation   ua1a_sb_stb.f--->ua1a_sb_stb.o
ua1a_sb_stb.f:
        rphiod:                               (...Block Software zum sb)
        rpbd1:                                (.....Block Dispatcher R.)
        rpbini:
        rpbfin:
        rpsini:
        rpsfin:
        rpo1ss:
        rpo1sf:
        .......                               (.....48 weitere Routinen)
        sm_1:                                 (...Server Stub zu ropen1)
        sm_2:                                 (...Server Stub zu rwrite1)
        sm_3:                                 (...Server Stub zu rclose1)
        sm_d1:                                (.....Module Dispatch. sm)
        sm_nam:
        sm_ini:
make module      ua1a_sb.o--->ua1a_sb
%
```

Anschließend sollte man Experiment 1 in der eigenen Directory dup_ua1a wiederholen, d.h. wie auf Seite 221 nur ohne die cd Kommandos am Anfang.

Das einzige *Geheimnis* ist noch der makefile. (Die Software wurde im Theorie-Kapitel erläutert.) Anstatt sich diesen im Detail klar zu machen, sollte man eher mit dem folgendes Experiment das Beispiel von Hand kompilieren und binden (z.B. mit f77, s.Seite 219).

Experiment 3:

```
% cd
% mkdir my_ua1a
% cd    my_ua1a
% cp $DFNRPCEXAMPLES/ua1a/ua1a.rpi       .
% cp $DFNRPCEXAMPLES/ua1a/ua1a_cb_stb.m4 .
% cp $DFNRPCEXAMPLES/ua1a/ua1a_sb_stb.m4 .

% $DFNRPCBIN/dfnrpcgen -T ieee ua1a
dfnrpcgen option type = -T
          option      = ieee
          application = ua1a

% f77  -o ua1a_lb  ua1a_lb.f
ua1a_lb.f:
 ... ...
```

```
% ln -s $DFNRPCINCLUDE/dfnrpc.inc dfnrpc.inc
% ln -s $DFNRPCINCLUDE/dfnrpc_stub.inc dfnrpc_stub.inc


% f77  -o ua1a_cb  ua1a_cb.f ua1a_cb_stb.f  $DFNRPCLIB/libdfnrpc.a
ua1a_cb.f:                                          (ggf. mit netlib)
  ... ...
ua1a_cb_stb.f:
  ... ...
Linking:

% f77  -o ua1a_sb  ua1a_sb.f ua1a_sb_stb.f  $DFNRPCLIB/libdfnrpc.a
ua1a_sb.f:                                          (ggf. mit netlib)
  ... ...
ua1a_sb_stb.f:
  ... ...
Linking:

%
```

In diesem Beispiel wurde jeder Block mit nur einem Compiler-Aufruf kompiliert und gebunden (netlib s. Tabelle Seite 218).

14.3.3 Beispiel ua1a - so einfach wie möglich

Im Beispiel *ua1a* enthält der Master Process den Client Block cb, bzw. der Slave Process den Server Block sb, wie in Abb. 4.13 dargestellt. Dies ist der *Fall 1* des Object Handling; die Routinen hierfür beginnen daher mit *RPO1*.

Der *automatische* Prozeßstart des Slaves ist bei ua1a darauf reduziert, daß die Object Handling Routinen des Master Process den Anwender auffordern, den Slave Process von Hand zu starten. Beim Master schreiben die Object Handling Routinen die für den Verbindungsaufbau nötigen Informationen auf den Bildschirm. Beim Slave werden diese Daten dann interaktiv von den Object Handling Routinen wieder eingelesen.

Die Object Handling Routinen des Beispiels ua1a seien gegenübergestellt:

```
C    C L I E N T  and  M A S T E R            | C    S E R V E R  and  S L A V E
                                              |
C*[MOD_BEG] cp, NAME='client special routines' | C*[MOD_BEG] sp,NAME='server spec.'
 SUBROUTINE RPCINI                            |  PROGRAM UA1SRV
  INCLUDE 'dfnrpc.inc'                        |   INCLUDE 'dfnrpc.inc'
  CHARACTER*40  Master_Host,       Slave_Host |   INTEGER      IRET,ISERID,IOBJID
  INTEGER   Master_Host_Length, Slave_Host_Length |
  CHARACTER*300 Object_Start_String           |
  INTEGER       Start_Flags, IRET             |
  CALL RPBINI (IRET)                          |   CALL RPBINI (IRET)
  CALL RPSINI(CT4TCP,CT4NO,CT4SO,1,1,ISERID,IRET) |   CALL RPSINI (CT4TCP,CT4NO,CT4SO,
  WRITE(*,*)'rpcini- please type Local Hostname' | +             1, 1, ISERID, IRET)
     READ (*,'(A)') Master_Host               |
     Master_Host_Length=INDEX(Master_Host,' ')-1 |
  WRITE(*,*)'rpcini- please type Remote Hostname' |
     READ (*,'(A)') Slave_Host                |
```

```
     Slave_Host_Length = INDEX(Slave_Host,' ')-1  |
  Start_Flags = OBJSSC + OBJSNC                   |
  Object_Start_String='Please start ua1*_sb on '  |
+          // Slave_Host(1:Slave_Host_Length)     |
  CALL RP01CS (Start_Flags,                       |   CALL RP01SS (IOBJID,IRET)
+            Master_Host(1:Master_Host_Length),   |
+            Object_Start_String, IOBJID, IRET)   |
  WRITE (*,*) ' rpcini - rpc initialized'         |   WRITE (*,*)
  RETURN                                          | + ' ua1srv - rpc initialized'
 END                                              |
                                                  |   WRITE (*,*)
                                                  | + ' ua1srv - starting dispatcher'
                                                  |   CALL RPBD1  (IRET)
                                                  |   WRITE (*,*)
                                                  | + ' ua1srv - dispatcher executed'
 SUBROUTINE RPCFIN                                |
  INCLUDE 'dfnrpc.inc'                            |
  INTEGER IRET                                    |
  CALL RP01CF (IRET)                              |   CALL RP01SF (IRET)
  CALL RPSFIN (IRET)                              |   CALL RPSFIN (IRET)
  CALL RPBFIN (IRET)                              |   CALL RPBFIN (IRET)
  WRITE (*,*) ' rpcfin - rpc finished'            |   WRITE (*,*)
                                                  | + ' ua1srv - rpc finished'
  RETURN                                          |  STOP
 END                                              | END
C*[MOD_END]                                       | C*[MOD_END]
```

Im folgenden werden die einzelnen Statements erklärt.

INCLUDE 'dfnrpc.inc' definiert die DFN-RPC Enumerations, siehe Kapitel 6.6 auf Seite 155.

CALL RPBINI (IRET) initialisiert einen Block.

Out	IRET = 0	erfolgreich ausgeführt

CALL RPSINI(CT4TCP,CT4N0,CT4S0,1,1,ISERID,IRET) initialisiert einen Transport Service.

In	CT4TCP	Transport Service: TCP
In	CT4N0	Network Service: Default
In	CT4S0	Transport Software: Default
In	1	Maximum der Verbindungen (incoming)
In	1	Maximum der Verbindungen (outgoing)
Out	ISERID	Service ID – hier ohne Bedeutung
Out	IRET = 0	erfolgreich ausgeführt

Start_Flags = OBJSSC + OBJSNC legt die Art des Objekt Starts fest.

In	OBJSSC	OBJect Start Flag – Slave Csh d.h. daß der Slave in einer C-Shell laufen wird
In	OBJSNC	OBJect Start Flag – No Call d.h. daß mit nachfolgendem Object_Start_String kein *CALL SYSTEM* gemacht wird, sondern daß dieser von der Object Start Routine *RPO1CS* nur auf den Bildschirm (stdout) geschrieben wird, um den Anwender aufzufordern, den Slave von Hand (interaktiv) zu starten

Object_Start_String='Please start ua1$*$_sb on ' // Slave_Host(1:Slave_Host_Length)

s.o. Start_Flags

CALL RPO1CS (Start_Flags, Master_Host(1:Master_Host_Length), Object_Start_String, IOBJID, IRET)

Mit diesem Aufruf wird das entfernte Object, d.h. der Slave, bzw. der Prozeß mit dem Server_Side Block sb, gestartet. Da das OBJSNC Flag gesetzt ist, wird diese Routine real den Slave nicht starten, sondern nur den Anwender auffordern dies interaktiv selbst zu tun. Beim Slave ist als Gegenstück *OBJ1SS* aufzurufen.

In	Start_Flags	s.o.
In	Master_Host	Die für die RPCs nötige Transport-Verbindung wird vom Slave aus zu diesem Master-Prozeß aufgebaut. Hierfür muß der Slave die Netzwerk-Adresse des Master Prozeß wissen. Da hier das No_Call Flag OBJSNC gesetzt ist, wird der Master_Host Parameter (zusammen mit den Start_Flags und dem für diese Verbindung generierten Transport Service Access Point, kurz TSAP) nur auf den Bildschirm geschrieben, damit der Anwender diese Daten beim von Hand gestarteten Slave-Prozeß wieder eingeben kann.
In	Object_Start_String	s.o.
Out	IOBJID	Object Id - hier ohne Bedeutung
Out	IRET = 0	erfolgreich ausgeführt

CALL RPO1SS (IOBJID, IRET) - Gegenstück zu RPO1CS beim Slave.

Out	IOBJID	Object Id - hier ohne Bedeutung
Out	IRET = 0	erfolgreich ausgeführt

CALL RPBD1 (IRET) - Aufruf der Block Dispatcher Routine

Out	IRET = 0	erfolgreich ausgeführt

Der Block Dispatcher empfängt server-seitig die Datenpakete, die die Client-Stubs abschicken, wertet die Header-Information aus und ruft die entsprechende Server-Stub Routine auf. Diese ruft dann die eigentliche Anwendungs-Routine, die mittels Remote Procedure Call gerufen wurde. Der Block Dispatcher bleibt aktiv, bis client-seitig RPO1CF gerufen wird.

CALL RPO1CF (IRET) - Object Finish Routine beim Master/Client.

Out	IRET = 0	erfolgreich ausgeführt

CALL RPO1CF (IRET) - Object Finish Routine beim Slave/Server.

Out	IRET = 0	erfolgreich ausgeführt

CALL RPO1CF (IRET) - Transport Service Finish.

Out	IRET = 0	erfolgreich ausgeführt

CALL RPO1CF (IRET) - Block Finish.

Out	IRET = 0	erfolgreich ausgeführt

14.4 Das vollständige Beispiel ua1a

```
C*[BLK_OPT] MASTER_STATIC_AUTHORIZATION = 124124123
C*[BLK_OPT] SLAVE_STATIC_AUTHORIZATION = 111079698
C*[MOD_PAR] sm ,DEPTH=0 ,CODE=4


C*[BLK_BEG] cb, NAME='ua1 client block', DEPTH=0, CLIENT_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     cp, LOCAL
C*[MOD_USE]     sm, CLIENT
C*[BLK_END]

C*[BLK_BEG] sb, NAME='ua1 server block', DEPTH=0, SERVER_SITE
C*[MOD_USE]     sp, LOCAL
C*[MOD_USE]     sm, SERVER
C*[BLK_END]
```

```
C*[BLK_BEG] lb, NAME='ua1 local block', DEPTH=0, LOCAL_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     sm, LOCAL
C*[BLK_END]

C*[MOD_BEG] cm, NAME='ua1 client module - main program'

C
      program rpitst
      character*10 filestring
      character*7 statusstring
      integer lun
      character*12 formatstr
      integer satzlaenge
      parameter(dateilaenge=20,satzlaenge=4)
      integer iein(satzlaenge)
      real    rein(satzlaenge)
C----------------------------------------------------
C*RPC call rpcini
C----------------------------------------------------
      filestring='ua1.dat'
      statusstring='unknown'
      lun=10
      call ropen1 (lun,filestring
     +              ,statusstring
     +                                  )
C----------------------------------------------------
      formatstr='(4(i5,f5.1))'
      num = 0
      do 100 i=1,dateilaenge
        do 50 j=1,satzlaenge
          num = num + 1
          iein(j) = num
          rein(j) = num
  50    continue
        call rwrite1 (lun,formatstr,iein,rein,satzlaenge)
 100  continue
C----------------------------------------------------
      call rclose1 (lun)
C----------------------------------------------------
C*RPC call rpcfin
C----------------------------------------------------
      stop
      end
C----------------------------------------------------
C*[MOD_END]

C
C*[MOD_BEG] cp, NAME='ua1 client rpc special routines'

      SUBROUTINE RPCINI

        INCLUDE 'dfnrpc.inc'
```

```
        CHARACTER*40  Master_Host,        Slave_Host
        INTEGER       Master_Host_Length, Slave_Host_Length
        CHARACTER*300 Object_Start_String
        INTEGER       Start_Flags, IRET

        CALL RPBINI (IRET)
           IF (IRET.NE.0) GOTO 98
        CALL RPSINI (CT4TCP, CT4NO, CT4SO, 1, 1, ISERID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - please type Local Hostname'
           READ (*,'(A)')         Master_Host
           Master_Host_Length = INDEX(Master_Host,' ')-1
        WRITE (*,*) ' rpcini - please type Remote Hostname'
           READ (*,'(A)')         Slave_Host
           Slave_Host_Length = INDEX(Slave_Host,' ')-1
        Start_Flags = OBJSSC + OBJSNC
        Object_Start_String =  'Please start ua1*_sb on '
     +                         // Slave_Host(1:Slave_Host_Length)
        CALL RPO1CS (Start_Flags, Master_Host(1:Master_Host_Length),
     +               Object_Start_String, IOBJID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - rpc correctly initialized'
       RETURN
98    CONTINUE
        WRITE (*,*) ' rpcini - error in rpc initialization causes stop'
       STOP
      END

      SUBROUTINE RPCFIN

        INCLUDE 'dfnrpc.inc'

        INTEGER IRET

        CALL RPO1CF (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' rpcfin - rpc correctly finished'
       RETURN
99    CONTINUE
        WRITE (*,*) ' rpcfin - error in rpc finishing causes stop'
       STOP
      END
C*[MOD_END]


C*[MOD_BEG] sp, NAME='ua1 server rpc special program'

      PROGRAM UA1SRV

        INCLUDE 'dfnrpc.inc'
```

```
        INTEGER       IRET, IOBJID

        CALL RPBINI (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSINI (CT4TCP, CT4NO, CT4S0, 1, 1, ISERID, IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPO1SS (IOBJID,IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly initialized'

        WRITE (*,*) ' ua1srv - starting rpc main dispatcher'
        CALL RPBD1  (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc main dispatcher correctly executed'

        CALL RPO1SF (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly finished'
       STOP

99    CONTINUE
        WRITE (*,*) ' ua1srv - server stopped because of rpc error'
       STOP
      END
C*[MOD_END]
C
C*[MOD_BEG] sm, NAME='ua1 server routines'
C*[RPI_BEG] ASYNCHRONOUS
      subroutine ropen1 (lun,filestring,statusstring)

      character*(*) filestring
C*RPC+                           [IN]
      character*(*) statusstring
C*RPC+                           [IN]
      integer lun
C*RPC+                           [IN]
C*[RPI_NON]
C     open(lun,file=filestring,status=statusstring)
      write (*,'(a,i3,1x,a,1x,a)')
     +         ' open ',lun,filestring,statusstring
      return
      end
C*[RPI_END]

C---------------------------------------------------
C*[RPI_BEG]
      subroutine rwrite1 (lun,formatstr,iwrite,rwrite,satzlaenge)
      character*(*) formatstr
C*RPC+                           [IN]
```

```
      integer satzlaenge
C*RPC+                        [IN]
      integer lun
C*RPC+                        [IN]
      integer iwrite(satzlaenge)
C*RPC+                        [IN]
      real    rwrite(satzlaenge)
C*RPC+                        [IN]
C.....alternative, falls die satzlaenge auch 0 sein darf:
C     integer iwrite(*)
C RPC+                        [IN, MAXINDEX=satzlaenge]
C     real    rwrite(*)
C RPC+                        [IN, MAXINDEX=satzlaenge]
C*[RPI_NON]
C     write(lun,formatstr) (iwrite(i),rwrite(i),i=1,satzlaenge)
      write(*,'(a,i3,a,10(i5,f5.1))') ' write',lun,formatstr,
     +                    (iwrite(i),rwrite(i),i=1,satzlaenge)
      return
      end
C*[RPI_END]
C-----------------------------------------------------
C*[RPI_BEG]
      subroutine rclose1 (lun)
      integer lun
C*RPC+             [IN]
C*[RPI_NON]
C     close(lun)
      write(*,'(a,i3)') ' close',lun
      return
      end
C*[RPI_END]
C*[MOD_END]
```

14.5 Besonderheiten bei mehreren Servern zu einem Client

Benutzt ein Client mehrere Server so können die Server-Blöcke (Objekte) zur gleichen Objektklasse gehören, aber auch unterschiedlich sein. Beim Starten jedes Server-Objekts (Blocks) bekommt es beim Client einen Objekt-Identifier (OBJID). Die Objekt-Start-Routine speichert diesen Wert in der internen globalen Variablen **Actual_Object_Identifier** ab. Alle (nachfolgenden) RPCs beziehen sich auf diesen Actual_Object_Identifier. Will man ein anderes Objekt ansprechen, z.B. ein zuvor gestartetes, dann muß man mit **CALL RPOUSE** den Actual_Object_Identifier auf ein anderes Objekt umsetzen.

Wenn man dies vergißt, tritt typischerweise der Fehler auf, daß eine Routine in dem aktuellen Objekt nicht gefunden werden kann.

Wenn man zudem die Server mit verschiedenen Transport Services erreichen will, dann muß man beachten, daß eine Objekt-Start-Routine sich immer auf den **aktuellen Service**, d.h. den zuletzt geöffneten bezieht. Wenn man sich auf einen früher geöffneten Service beziehen möchte, dann muß man RPSUSE aufrufen. Mit RPSUSE legt man den „aktuellen Service“ neu fest.

Zum Umschalten zwischen schon gestarteten Objekten benötigt man nur RP**O**USE, RP**S**USE wird nur beim Objekt-Start benötigt.

14.6 Transportdienstverwaltung

```
----------------------------------------------------------------
| Procedure      RPOGSR                                        |
|                                                              |
| Synopsis       CALL RPOGSR (LL_SND, LL_REC, LL_USE,          |
|                             LR_SND, LR_REC, LR_USE, IRET)    |
|                                                              |
| Task           Get Send/Receive Buffer Length                |
|                                                              |
| Parameter       Type           I/O Usage                     |
|                                     Local_Site:              |
| LL_SND          integer        Out Length of the Send Buffer|
| LL_REC          integer        Out Length of Receive  Buffer|
| LL_USE          integer        Out Used Part of Receive Buf.|
|                                     Remote_Site:             |
| LR_SND          integer        Out Length of the Send Buffer|
| LR_REC          integer        Out Length of Receive  Buffer|
| LR_USE          integer        Out Used Part of Receive Buf.|
| IRET            integer        Out return code               |
|                                                              |
| Determination of LL_USE and LR_USE:                          |
| A package with LR_SND bytes is sent from remote to local and |
| received in the receive buffer (its length is LL_REC). It is |
| possible, that this package was devided into smaller ones,   |
| because there are smaller buffers in the underlaying protocol|
| stack. LL_USE is the length of the first portion received in |
| the local receive buffer.                                    |
| In the other direction LR_USE is the first portion received  |
| in the remote receive buffer.                                |
| All values are meassured in bytes.                           |
----------------------------------------------------------------
```

```
----------------------------------------------------------------
| Procedure      RPOSSR                                        |
|                                                              |
| Synopsis       CALL RPOSSR(LL_SND,LL_REC,LR_SND,LR_REC,IRET) |
| Task           Set Send/Receive Buffer Length                |
|                                                              |
| Parameter       Type           I/O Usage                     |
|                                     Local_Site:              |
| LL_SND          integer        In  Length of the Send Buffer|
| LL_REC          integer        In  Length of Receive  Buffer|
|                                     Remote_Site:             |
| LR_SND          integer        In  Length of the Send Buffer|
| LR_REC          integer        In  Length of Receive  Buffer|
| IRET            integer        Out return code               |
|                                                              |
| If the wished length is outside of the allowed range         |
| 600..65528, then the nearest bound is used.                  |
| The range is defined by S/R1B_MININST .. S/R1B_MAXINST.      |
| All values are meassured in bytes.                           |
----------------------------------------------------------------
```

14.7 Server-Start ohne Restart, Master enthält Client

```
----------------------------------------------------------------
| Procedure      RPO1CS                                        |
|                                                              |
| Synopsis       CALL (Start_Flags, Master_Host,               |
|                      Object_Start, OBJID, IRET)              |
|                                                              |
| Task           Initialization of an object                   |
|                                                              |
| Argument        Type           I/O Usage                     |
|                                                              |
| Start_Flags     integer        In  sum of following flags    |
|      - the local settings are for a master shell, type=      |
|        (unused in this subroutine, used only with restart)   |
|                 0               = not defined                |
|                 SFLG_Master_Sh  = sh                         |
|                 SFLG_Master_Csh = csh                        |
|      - the settings must be written for a slave shell, type= |
|                 0               = not defined                |
|                 SFLG_Slave_Sh   = sh                         |
|                 SFLG_Slave_Csh  = csh                        |
|      - write the settings on file dfnrpc_settings            |
|                 0               = no write (normal)          |
|                 SFLG_Write      = write                      |
|      - substitute string DFNRPC_SETTINGS in the Object_Start |
|         (//n will be always substituted in the Object_Start  |
|          by 'backslash backslash newline')                   |
|                 0               = substitute (normal)        |
|                 SFLG_No_Subst   = suppress substitution      |
|      - insert some 'backslash backslash newline' while subst.|
|                 0               = insert (normal)            |
|                 SFLG_No_Backslash= suppress insertation      |
|      - make additionally an interactive authorization test   |
|                 0               = no interactive a. (normal)|
|                 SFLG_Int_Auth   = interactive authorization |
|      - make additionally a dynamic authorization test        |
|                 0               = no dynamic a. (normal)     |
|                 SFLG_Dyn_Auth   = dynamic authorization      |
|      - make system call to start the remote object           |
|                 0               = make system call (normal) |
|                 SFLG_No_Call    = suppress system call       |
|      - make control output to stdout                         |
|                 0               = make output (normal)       |
|                 SFLG_Trace_Obj  = additional trace output    |
|                                                              |
| Master_Host     string         In  Local Hostname,           |
|                                    callable by the slave     |
| Object_Start    string         In  text for interac. request|
|                                    or input for system call |
| OBJID           integer        Out object identifier         |
| IRET            integer        Out return code               |
----------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RP01SS                                       |
|                                                             |
| Synopsis       CALL (OBJID, IRET)                           |
|                                                             |
| Task           Initialization of an object                  |
|                                                             |
| Argument        Type            I/O Usage                   |
| OBJID           integer         Out object identifier       |
| IRET            integer         Out return code             |
|                                                             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RP01CF                                       |
|                                                             |
| Synopsis       CALL (IRET)                                  |
|                                                             |
| Task           Finish of an object                          |
|                                                             |
| Argument        Type            I/O Usage                   |
| IRET            integer         Out return code             |
|                                                             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RP01SF                                       |
|                                                             |
| Synopsis       CALL (IRET)                                  |
|                                                             |
| Task           Finish of an object                          |
|                                                             |
| Argument        Type            I/O Usage                   |
| IRET            integer         Out return code             |
|                                                             |
---------------------------------------------------------------
```

14.8 Server-Start mit Restart, Master enthält Client

```
-----------------------------------------------------------------
| Procedure      RPO2CI                                         |
|                                                               |
| Synopsis       CALL (Start_Flags, Master_Host,                |
|                      Slave_Host, Object_Start, IRET)          |
|                                                               |
| Task           Initialization of an object                   |
|                                                               |
| Argument        Type            I/O Usage                     |
| Start_Flags     integer         In  sum of following flags    |
|      - the local settings are for a master shell, type=       |
|        (unused in this subroutine, used only with restart)    |
|                 0                = not defined                |
|                 SFLG_Master_Sh   = sh                         |
|                 SFLG_Master_Csh  = csh                        |
|      - the settings must be written for a slave shell, type=  |
|                 0                = not defined                |
|                 SFLG_Slave_Sh    = sh                         |
|                 SFLG_Slave_Csh   = csh                        |
|      - write the settings on file dfnrpc_settings             |
|                 0                = no write (normal)          |
|                 SFLG_Write       = write                      |
|      - substitute string DFNRPC_SETTINGS in the Object_Start  |
|         (//n will be always substituted in the Object_Start   |
|          by 'backslash backslash newline')                    |
|                 0                = substitute (normal)        |
|                 SFLG_No_Subst    = suppress substitution      |
|      - insert some 'backslash backslash newline' while subst.|
|                 0                = insert (normal)            |
|                 SFLG_No_Backslash= suppress insertation       |
|      - make additionally an interactive authorization test    |
|                 0                = no interactive a. (normal)|
|                 SFLG_Int_Auth    = interactive authorization  |
|      - make additionally a dynamic authorization test         |
|                 0                = no dynamic a. (normal)     |
|                 SFLG_Dyn_Auth    = dynamic authorization      |
|      - make system call to start the remote object            |
|                 0                = make system call (normal)  |
|                 SFLG_No_Call     = suppress system call       |
|      - make control output to stdout                          |
|                 0                = make output (normal)       |
|                 SFLG_Trace_Obj   = additional trace output    |
|                                                               |
| Master_Host     string          In  Local Hostname, callable  |
|                                     by the slave, used only   |
|                                     if SFLG_No_Call not set   |
| Slave_Host      string          In  Remote Hostname           |
| Object_Start    string          In  text for interac. request|
|                                     or input for system call  |
| IRET            integer         Out return code               |
-----------------------------------------------------------------
```

```
----------------------------------------------------------------
|                                                              |
| Procedure      RPO2SI                                        |
|                                                              |
| Synopsis       CALL (OBJID, IRET)                            |
|                                                              |
| Task           Initialization of an object                   |
|                                                              |
| Argument        Type           I/O Usage                     |
| OBJID           integer        Out object identifier         |
| IRET            integer        Out return code               |
|                                                              |
----------------------------------------------------------------
```

```
----------------------------------------------------------------
|                                                              |
| Procedure      RPO2CS                                        |
|                                                              |
| Synopsis       CALL (OBJID, IRET)                            |
|                                                              |
| Task           Initialization of an object                   |
|                                                              |
| Argument        Type           I/O Usage                     |
| OBJID           integer        Out object identifier         |
| IRET            integer        Out return code               |
|                                                              |
----------------------------------------------------------------
```

```
----------------------------------------------------------------
|                                                              |
| Procedure      RPO2SS                                        |
|                                                              |
| Synopsis       CALL (IRET)                                   |
|                                                              |
| Task           Initialization of an object                   |
|                                                              |
| Argument        Type           I/O Usage                     |
| IRET            integer        Out return code               |
----------------------------------------------------------------
```

```
----------------------------------------------------------------
|                                                              |
| Procedure      RPO2CF                                        |
|                                                              |
| Synopsis       CALL (Restart_Flag, IRET)                     |
|                                                              |
| Task           Finish of an object                           |
|                                                              |
| Argument        Type           I/O Usage                     |
| Restart_Flag    integer        In  0=Restart_Flag_NO, 1=YES |
| IRET            integer        Out return code               |
----------------------------------------------------------------
```

```
----------------------------------------------------------------
|                                                              |
| Procedure      RPO2SF                                        |
|                                                              |
| Synopsis       CALL RPO2SF (Restart_Flag, IRET)              |
|                                                              |
| Task           Finish of an object                           |
|                                                              |
| Argument        Type            I/O Usage                    |
| Restart_Flag    integer         Out 0=Restart_Flag_NO, 1=YES |
| IRET            integer         Out return code              |
|                                                              |
----------------------------------------------------------------
```

14.9 Object-Start and Finish, Master enthält Server

```
-----------------------------------------------------------------
|                                                               |
| Procedure      RPO3SI                                         |
|                                                               |
| Synopsis       CALL (Start_Flags, Master_Host,                |
|                      Object_Start, OBJID, IRET)               |
|                                                               |
| Task           Initialization of an object                    |
|                                                               |
| Argument        Type            I/O Usage                     |
| Start_Flags     integer         In  sum of following flags    |
|      - the local settings are for a master shell, type=       |
|        (unused in this subroutine, used only with restart)    |
|                 0                = not defined                |
|                 SFLG_Master_Sh   = sh                         |
|                 SFLG_Master_Csh  = csh                        |
|      - the settings must be written for a slave shell, type=  |
|                 0                = not defined                |
|                 SFLG_Slave_Sh    = sh                         |
|                 SFLG_Slave_Csh   = csh                        |
|      - write the settings on file dfnrpc_settings             |
|                 0                = no write (normal)          |
|                 SFLG_Write       = write                      |
|      - substitute string DFNRPC_SETTINGS in the Object_Start  |
|         (//n will be always substituted in the Object_Start   |
|          by 'backslash backslash newline')                    |
|                 0                = substitute (normal)        |
|                 SFLG_No_Subst    = suppress substitution      |
|      - insert some 'backslash backslash newline' while subst.|
|                 0                = insert (normal)            |
|                 SFLG_No_Backslash= suppress insertation       |
|      - make additionally an interactive authorization test    |
|                 0                = no interactive a. (normal)|
|                 SFLG_Int_Auth    = interactive authorization |
|      - make additionally a dynamic authorization test         |
|                 0                = no dynamic a. (normal)     |
|                 SFLG_Dyn_Auth    = dynamic authorization      |
|      - make system call to start the remote object            |
|                 0                = make system call (normal) |
|                 SFLG_No_Call     = suppress system call       |
|      - make control output to stdout                          |
|                 0                = make output (normal)       |
|                 SFLG_Trace_Obj   = additional trace output    |
|                                                               |
| Master_Host     string          In  Local Hostname,           |
|                                     callable by the slave     |
| Object_Start    string          In  text for interac. request|
|                                     or input for system call |
| OBJID           integer         Out object identifier         |
| IRET            integer         Out return code               |
-----------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RPO3SS                                       |
|                                                             |
| Synopsis       CALL (IRET)                                  |
|                                                             |
| Task           Initialization of an object                  |
|                                                             |
| Argument        Type            I/O Usage                   |
| IRET            integer         Out return code             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RPO3CS                                       |
|                                                             |
| Synopsis       CALL (OBJID, IRET)                           |
|                                                             |
| Task           Initialization of an object                  |
|                                                             |
| Argument        Type            I/O Usage                   |
| OBJID           integer         Out object identifier       |
| IRET            integer         Out return code             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RPO3CF                                       |
|                                                             |
| Synopsis       CALL (Restart_Flag, IRET)                    |
|                                                             |
| Task           Finish of an object                          |
|                                                             |
| Argument        Type            I/O Usage                   |
| Restart_Flag    integer         In  0=Restart_Flag_NO       |
|                                     1=Restart_Flag_YES      |
| IRET            integer         Out return code             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
| Procedure      RPO3SF                                       |
|                                                             |
| Synopsis       CALL (Restart_Flag, IRET)                    |
|                                                             |
| Task           Finish of an object                          |
|                                                             |
| Argument        Type            I/O Usage                   |
| Restart_Flag    integer         Out 0=Restart_Flag_NO       |
|                                     1=Restart_Flag_YES      |
| IRET            integer         Out return code             |
---------------------------------------------------------------
```

14.10 Beispiel für interaktiven Objektstart (aus ua1a)

Das folgende Beispiel wird mitgeliefert in $DFNRPCEXAMPLES/ua1a. Es ist im User's Guide Kapitel 4.8 auf Seite 42 bereits ausführlich erläutert.

```
C*[MOD_BEG] cp, NAME='ua1 client rpc special routines'

      SUBROUTINE RPCINI

        INCLUDE 'dfnrpc.inc'

        CHARACTER*40  Master_Host,        Slave_Host
        INTEGER       Master_Host_Length, Slave_Host_Length
        CHARACTER*300 Object_Start_String
        INTEGER       Start_Flags, IRET

        CALL RPBINI (IRET)
           IF (IRET.NE.0) GOTO 98
        CALL RPSINI (CT4TCP, CT4NO, CT4SO, 1, 1, ISERID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - please type Local Hostname'
           READ (*,'(A)')         Master_Host
           Master_Host_Length = INDEX(Master_Host,' ')-1
        WRITE (*,*) ' rpcini - please type Remote Hostname'
           READ (*,'(A)')         Slave_Host
           Slave_Host_Length = INDEX(Slave_Host,' ')-1
        Start_Flags = OBJSSC + OBJSNC
        Object_Start_String =  'Please start ua1*_sb on '
     +                         // Slave_Host(1:Slave_Host_Length)
        CALL RPO1CS (Start_Flags, Master_Host(1:Master_Host_Length),
     +               Object_Start_String, IOBJID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - rpc correctly initialized'
       RETURN
98    CONTINUE
        WRITE (*,*) ' rpcini - error in rpc initialization causes stop'
       STOP
      END

      SUBROUTINE RPCFIN

        INCLUDE 'dfnrpc.inc'

        INTEGER IRET

        CALL RPO1CF (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' rpcfin - rpc correctly finished'
       RETURN
```

```
99    CONTINUE
        WRITE (*,*) ' rpcfin - error in rpc finishing causes stop'
       STOP
      END
C*[MOD_END]


C*[MOD_BEG] sp, NAME='ua1 server rpc special program'

      PROGRAM UA1SRV

        INCLUDE 'dfnrpc.inc'

        INTEGER       IRET, IOBJID

        CALL RPBINI (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSINI (CT4TCP, CT4NO, CT4SO, 1, 1, ISERID, IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPO1SS (IOBJID,IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly initialized'

        WRITE (*,*) ' ua1srv - starting rpc main dispatcher'
        CALL RPBD1  (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc main dispatcher correctly executed'

        CALL RPO1SF (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly finished'
       STOP

99    CONTINUE
        WRITE (*,*) ' ua1srv - server stopped because of rpc error'
       STOP
      END
C*[MOD_END]
```

14.11 Objektstart mit NQS (aus ua1b)

Das Beispiel ua1b in $DFNRPCEXAMPLES/ua1b unterscheidet sich von ua1a durch seine RPC-Spezialprogramme. Der Server von ua1b wird mit Hilfe des Network Queueing Systems (NQS) gestartet. Die Bedienung des Beispiels wird in Kap. 9 auf S. 165 beschrieben.

```
C*[MOD_BEG] cp, NAME='ua1 client rpc special routines'

C
```

```
      SUBROUTINE RPCINI

        INCLUDE 'dfnrpc.inc'

        CHARACTER*80  Input_Line
        INTEGER       Master_Host_Beg
        INTEGER       Master_Host_End
        INTEGER       Slave_Host_Beg, Slave_User_Beg, Slave_Shell_Beg
        INTEGER       Slave_Host_End, Slave_User_End, Slave_Shell_End
        INTEGER       Queue_Name_Beg, Queue_Name_End
        CHARACTER*300 Object_Start_String
        INTEGER       Start_Flags, ISERID, IRET

C       --- input information
        WRITE (*,*) ' rpcini - please type on ONE input line'
        WRITE (*,*) '          separated by one blank'
        WRITE (*,*) '          - your local hostname'
        WRITE (*,*) '                  (which is used to make a tcp'
        WRITE (*,*) '                    connection from your remote'
        WRITE (*,*) '                    host to this local host)'

        WRITE (*,*) '          - your remote hostname'
        WRITE (*,*) '                  (see /etc/hosts)'
        WRITE (*,*) '          - your remote username'
        WRITE (*,*) '                  (which are used to start the'
        WRITE (*,*) '                   slave on the remote host using'
        WRITE (*,*) '                   "qsub")'
        WRITE (*,*) '          - the shell used on the remote host'
        WRITE (*,*) '                  (see /etc/passwd there'
        WRITE (*,*) '                  possible values are csh and sh)'
        WRITE (*,*) '          - and the queue-name'
        WRITE (*,*) '                  (which is used to place a'
        WRITE (*,*) '                   request in the stated queue,'
        WRITE (*,*) '                   see for :'
        WRITE (*,*) '            Workstation : qstat -l|grep "type=" '
        WRITE (*,*) '            CRAY :        qstat -f|grep "QUEUE:" '

        WRITE (*,*) '          Example: my_workstation rusc2 '
     +          //  'my_username_on_rusc2 csh cray2'
        READ (*,'(A)') Input_Line
        Master_Host_Beg = 1
        Master_Host_End = INDEX(Input_line,' ')-1
        Slave_Host_Beg  = Master_Host_End + 2
        Slave_Host_End  = Slave_Host_Beg - 2 +
     +                      INDEX(Input_Line(Slave_Host_Beg:),' ')
        Slave_User_Beg  = Slave_Host_End + 2
        Slave_User_End  = Slave_User_Beg - 2 +
     +                      INDEX(Input_Line(Slave_User_Beg:),' ')
        Slave_Shell_Beg = Slave_User_End + 2
        Slave_Shell_End = Slave_Shell_Beg + 2
        Queue_Name_Beg  = Slave_Shell_End + 2
        Queue_Name_End  = Queue_Name_Beg - 2 +
     +                      INDEX(Input_Line(Queue_Name_Beg:),' ')
C       --- Flags: OBJSDA = dynamic authorization
```

```
C                 OBJSNB = no backslashes inserted into DFNRPC_SETTINGS
C                 OBJSSC = remote login shell is csh
C                 OBJSSS = remote login shell is sh
      Start_Flags = OBJSDA + OBJSNB
      IF (Input_Line(Slave_Shell_Beg:Slave_Shell_End).EQ.'csh') THEN
         Start_Flags = Start_Flags + OBJSSC
      ELSEIF(Input_Line(Slave_Shell_Beg:Slave_Shell_End).EQ.'sh ')THEN
         Start_Flags = Start_Flags + OBJSSS
      ELSE
         WRITE (*,*) ' rpcini - error in input line'
         WRITE (*,*) '         local  host  = ',
     +               Input_Line(Master_Host_Beg:Master_Host_End)
         WRITE (*,*) '         remote host  = ',
     +               Input_Line(Slave_Host_Beg:Slave_Host_End)
         WRITE (*,*) '         remote user  = ',
     +               Input_Line(Slave_User_Beg:Slave_User_End)
         WRITE (*,*) '         remote shell = ',
     +               Input_Line(Slave_Shell_Beg:Slave_Shell_End)
         WRITE (*,*) '         queue name   = ',
     +               Input_Line(Queue_Name_Beg:Queue_Name_End)
         GOTO 98
      ENDIF

C     --- In the following string, the dfn-rpc object start routine
C         RPO1CS will substitute the place holder DFNRPC_SETTINGS
C         by shell commands (separeted by ';') to set a few
C         environment variables from which the remote rpc object
C         start routine can read the following data:
C          - the start flags to make a corresponding start,
C          - the local hostname and tsap (port number) to
C            make the dfn-rpc connection to this host
C          - dynamically generated passwords for the authorization.
C         Then the following string is used in a system call.
      Object_Start_String=
     +          ' cd $HOME;'
     +         // '/bin/rm -f ua1b_nqs_job;'
     +         // ' sed -e "s/xxxx/DFNRPC_SETTINGS/"'
     +         // ' $DFNRPCEXAMPLES/ua1b/ua1b_nqs_input'
     +         // ' > ua1b_nqs_job;'
     +         // ' qsub -q '
     +         // Input_Line(Queue_Name_Beg:Queue_Name_End)
C    +         // ' -u '
C    +         // Input_Line(Slave_User_Beg:Slave_User_End)
     +         // ' ua1b_nqs_job'



C     --- rpc block init
      CALL RPBINI (IRET)
         IF (IRET.NE.0) GOTO 98
C     --- rpc service init, here tcp/ip
      CALL RPSINI (CT4TCP, CT4NO, CT4SO, 1, 1, ISERID, IRET)
         IF (IRET.NE.0) GOTO 98
      WRITE (*,*) 'Authorization check for user '
```

```
     +          // Input_Line(Slave_User_Beg:Slave_User_End)
     +          //  ' on '
     +          // Input_Line(Slave_Host_Beg:Slave_Host_End)

C       --- rpc start remote object
        CALL RPO1CS (Start_Flags,
     +               Input_Line(Master_Host_Beg:Master_Host_End),
     +               Object_Start_String, IOBJID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - rpc correctly initialized'
       RETURN
98    CONTINUE
        WRITE (*,*) ' rpcini - error in rpc initialization causes stop'
       STOP
      END

      SUBROUTINE RPCFIN

        INCLUDE 'dfnrpc.inc'

C       --- rpc finish remote object
        CALL RPO1CF (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service finish
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc block finish
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' rpcfin - rpc correctly finished'
       RETURN
99    CONTINUE
        WRITE (*,*) ' rpcfin - error in rpc finishing causes stop'
       STOP
      END

C*[MOD_END]




C*[MOD_BEG] sp, NAME='ua1 server rpc special program'


      PROGRAM UA1SRV
C
        INCLUDE 'dfnrpc.inc'
C
        INTEGER       ISERID, IRET
C
C       --- rpc block init
        CALL RPBINI (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service init, here tcp/ip
```

```
        CALL RPSINI (CT4TCP, CT4N0, CT4S0, 1, 1, ISERID, IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc start this object
        CALL RPO1SS (IOBJID,IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly initialized'
C
C       --- rpc block dispatcher
        CALL RPBD1  (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc main loop correctly executed'
C
C       --- rpc finish this object
        CALL RPO1SF (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service finish
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc block finish
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly finished'

        STOP
C
99    CONTINUE
        WRITE (*,*) ' ua1srv - server stopped because of rpc error'
        STOP
      END
C*[MOD_END]
```

14.12 Objektstart mit Remote Shell (aus ua1c)

Das Beispiel ua1c in $DFNRPCEXAMPLES/ua1c unterscheidet sich von ua1a durch seine RPC-Spezialprogramme. Für den Start des ua1c Servers wird das Remote Shell Programm (rsh) verwendet. Die Bedienung des Beispiels wird in Kap. 9 auf S. 165 beschrieben.

```
C*[MOD_BEG] cp, NAME='ua1 client rpc special routines'

C
      SUBROUTINE RPCINI

        INCLUDE 'dfnrpc.inc'

        CHARACTER*80  Input_Line
        INTEGER       Master_Host_Beg
        INTEGER       Master_Host_End
        INTEGER       Slave_Host_Beg, Slave_User_Beg, Slave_Shell_Beg
        INTEGER       Slave_Host_End, Slave_User_End, Slave_Shell_End
        CHARACTER*300 Object_Start_String
        INTEGER       Start_Flags, ISERID, IRET
```

```
C       --- input information
        WRITE (*,*) ' rpcini - please type on ONE input line'
        WRITE (*,*) '          separated by one blank'
        WRITE (*,*) '          - your local hostname'
        WRITE (*,*) '                  (which is used to make a tcp'
        WRITE (*,*) '                    connection from your remote'
        WRITE (*,*) '                    host to this local host)'
        WRITE (*,*) '          - your remote hostname'
        WRITE (*,*) '          - your remote username'
        WRITE (*,*) '                  (which are used to start the'
        WRITE (*,*) '                   slave on the remote host using'
        WRITE (*,*) '                   the remote shell utility rsh)'
        WRITE (*,*) '          - and the shell used on the remote host'
        WRITE (*,*) '                  (see /etc/passwd there'
        WRITE (*,*) '                  possible values are csh and sh)'
        WRITE (*,*) '          Example: rusc2 my_workstation '
     +          //  'my_username_on_my_workstation csh'
        READ (*,'(A)') Input_Line
        Master_Host_Beg = 1
        Master_Host_End = INDEX(Input_line,' ')-1
        Slave_Host_Beg  = Master_Host_End + 2
        Slave_Host_End  = Slave_Host_Beg - 2 +
     +                     INDEX(Input_Line(Slave_Host_Beg:),' ')
        Slave_User_Beg  = Slave_Host_End + 2
        Slave_User_End  = Slave_User_Beg - 2 +
     +                     INDEX(Input_Line(Slave_User_Beg:),' ')
        Slave_Shell_Beg = Slave_User_End + 2
        Slave_Shell_End = Slave_Shell_Beg + 2
C       --- Flags: OBJSDA = dynamic authorization
C                  OBJSNB = no backslashes inserted into DFNRPC_SETTINGS
C                  OBJSSC = remote login shell is csh
C                  OBJSSS = remote login shell is sh
        Start_Flags = OBJSDA + OBJSNB
        IF (Input_Line(Slave_Shell_Beg:Slave_Shell_End).EQ.'csh') THEN
           Start_Flags = Start_Flags + OBJSSC
        ELSEIF(Input_Line(Slave_Shell_Beg:Slave_Shell_End).EQ.'sh ')THEN
           Start_Flags = Start_Flags + OBJSSS
        ELSE
           WRITE (*,*) ' rpcini - error in input line'
           WRITE (*,*) '          local  host  = ',
     +                 Input_Line(Master_Host_Beg:Master_Host_End)
           WRITE (*,*) '          remote host  = ',
     +                 Input_Line(Slave_Host_Beg:Slave_Host_End)
           WRITE (*,*) '          remote user  = ',
     +                 Input_Line(Slave_User_Beg:Slave_User_End)
           WRITE (*,*) '          remote shell = ',
     +                 Input_Line(Slave_Shell_Beg:Slave_Shell_End)
           GOTO 98
        ENDIF

C       --- In the following string, the dfn-rpc object start routine
C           RPO1CS will substitute the place holder DFNRPC_SETTINGS
C           by shell commands (separeted by ';') to set a few
C           environment variables from which the remote rpc object
```

```
C           start routine can read the following data:
C            - the start flags to make a corresponding start,
C            - the local hostname and tsap (port number) to
C              make the dfn-rpc connection to this host
C            - dynamically generated passwords for the authorization.
C           Then the following string is used in a system call.
        Object_Start_String =
     +          'rsh '
     +          // Input_Line(Slave_Host_Beg:Slave_Host_End)
     +          // ' -l ' // Input_Line(Slave_User_Beg:Slave_User_End)
     +          // ' -n ''DFNRPC_SETTINGS; cd $DFNRPCEXAMPLES/ua1c;'
     +          // ' ua1c_sb'' &'

C       --- rpc block init
        CALL RPBINI (IRET)
           IF (IRET.NE.0) GOTO 98
C       --- rpc service init, here tcp/ip
        CALL RPSINI (CT4TCP, CT4NO, CT4SO, 1, 1, ISERID, IRET)
           IF (IRET.NE.0) GOTO 98
C       --- rpc start remote object
        CALL RPO1CS (Start_Flags,
     +               Input_Line(Master_Host_Beg:Master_Host_End),
     +               Object_Start_String, IOBJID, IRET)
           IF (IRET.NE.0) GOTO 98
        WRITE (*,*) ' rpcini - rpc correctly initialized'
       RETURN
98    CONTINUE
        WRITE (*,*) ' rpcini - error in rpc initialization causes stop'
       STOP
      END

      SUBROUTINE RPCFIN

        INCLUDE 'dfnrpc.inc'

C       --- rpc finish remote object
        CALL RPO1CF (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service finish
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc block finish
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' rpcfin - rpc correctly finished'
       RETURN
99    CONTINUE
        WRITE (*,*) ' rpcfin - error in rpc finishing causes stop'
       STOP
      END

C*[MOD_END]
```

```
C*[MOD_BEG] sp, NAME='ua1 server rpc special program'


      PROGRAM UA1SRV
C
        INCLUDE 'dfnrpc.inc'
C
        INTEGER       ISERID, IRET
C
C       --- rpc block init
        CALL RPBINI (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service init, here tcp/ip
        CALL RPSINI (CT4TCP, CT4N0, CT4S0, 1, 1, ISERID, IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc start this object
        CALL RPO1SS (IOBJID,IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly initialized'
C
C       --- rpc block dispatcher
        CALL RPBD1  (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc main loop correctly executed'
C
C       --- rpc finish this object
        CALL RPO1SF (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc service finish
        CALL RPSFIN (IRET)
          IF (IRET.NE.0) GOTO 99
C       --- rpc block finish
        CALL RPBFIN (IRET)
          IF (IRET.NE.0) GOTO 99
        WRITE (*,*) ' ua1srv - rpc correctly finished'

        STOP
C
99    CONTINUE
        WRITE (*,*) ' ua1srv - server stopped because of rpc error'
        STOP
      END
C*[MOD_END]
```

14.13 Die Benutzeranwendung ua1

Die Anwendung ua1 wird in mehreren Variationen mitgeliefert, die unterschiedliche Techniken und Dienste zum Programmstart einsetzen. Die Anwendung selbst führt keinen Dialog.

14.13.1 Beispiel ua1a

Der Austausch von Informationen zum Verbindungsaufbau und der Server-Start geschehen durch geführten Dialog der Client- und Server-Programme. Das Client-Programm ua1a_cb muß zuerst gestartet werden. Es beginnt dann den Dialog. Der Benutzer beantwortet Fragen und erhält Anweisungen zum Start des Server-Programms ua1a_sb und zur Übergabe notwendiger Angaben zum Verbindungsaufbau. Der Server verbindet dann zum Client.

Für diese Vorgehensweise benötigt der Benutzer zwei Fenster oder Bildschirme zur Kontrolle der Programme des jeweiligen Rechners. Da das Beispiel ua1a die Grundlage für das Benutzerhandbuch ist, werden Programmaufbau und Dialogablauf hier nicht weiter erläutert.

14.13.2 Beispiel ua1b mit NQS

Das Beispiel ua1b benutzt eine vom RUS modifizierte Version des Network Queueing Systems (NQS). Die Modifikation besteht darin, daß eine Passwortabfrage integriert wurde. Dadurch ist ein sicherer Rechenzentrumsbetrieb gewährleistet. Dieses Programmsystem muß vom Systemverwalter auf beiden beteiligten Rechnern installiert werden. Im lokalen Home-Directory des Benutzers muß eine Datei .bhosts mit dem Eintrag

```
Rechneradresse Benutzername
```

gültig für den entfernten Rechner angelegt werden.

Das Client-Programm ua1b_cb erfragt nach dem Start eine Eingabezeile für die Serverstart-Informationen:

```
 rpcini - please type on ONE input line
          separated by one blank
          - your local hostname
                  (which is used to make a tcp
                    connection from your remote
                    host to this local host)
          - your remote hostname
                  (see /etc/hosts)
          - your remote username
                  (which are used to start the
                   slave on the remote host using
                   "qsub")
          - the shell used on the remote host
                  (see /etc/passwd there
                  possible values are csh and sh)
          - and the queue-name
                  (which is used to place a
                   request in the stated queue,
                   see for :
            Workstation : qstat -l|grep "type="
            CRAY :        qstat -f|grep "QUEUE:"
          Example: my_workstation rusc2 my_username_on_rusc2 csh rusc2
```

Nun kann eine Eingabezeile der Form

```
local_hostname remote_hostname remote_username remote_loginshell queue_name
```

mit der beispielhaften Kombination

```
local_hostname    = ruscvx
remote_hostname   = rusc2 (Langform, falls notwendig)
remote_username   = Username auf entferntem Rechner
remote_loginshell = csh oder sh (wie Eintrag in /etc/passwd !)
queue_name        = batch
```

eingegeben werden.

Der queue_name muß im NQS des local_host existieren. Das Client-Programm baut aus dieser Information automatisch einen Job auf und schickt ihn mit dem Programm qsub ab. Das vom RUS modifizierte Programm qsub führt dann eine Passwortabfrage und -verifikation durch.

Zuvor kann das NQS mit folgendem Beispiel getestet werden:

```
Inhalt der Datei 'job_file':
# QSUB
setenv HHH "Hallo"
echo " $HHH hier ist der Batchjob "
echo " Ende des Batchjobs job_file "

Start:
qsub -q <queue_name> job_file

Beobachten der Queues:
qstat -a

Listing und Fehlerprotokoll findet man dann lokal
im Homedirectory:

       job_file.onnnn
       job_file.ennnn
```

14.13.3 Beispiel ua1c mit rsh

Das Beispiel ua1c benützt das Remote Shell Programm rsh, um den Server-Prozess zu starten.

Für den Einsatz von rsh müssen folgende Vorbereitungen getroffen werden:

- local_hostname und local_username in Datei .rhosts im Home-Directory des entfernten Rechners eintragen
- Auf entferntem Rechner gegebenfalls rshd daemon starten
- Testkommando: rsh remote_hostname -l remote_username ls
- Da rsh auf dem entfernten Rechner keine Login-Prozedur durchläuft, sollte man mit folgendem Kommando testen, ob das Server-Programm ua1c_sb auch gefunden wird:

```
rsh remote_hostname 'cd $DFNRPCEXAMPLES/ua1c; pwd; ls ua1c_sb'
```

Nach dem Start fragt das Client-Programm ua1c_cb nach der Eingabezeile:

```
 rpcini - please type on ONE input line
         separated by one blank
         - your local hostname
                 (which is used to make a tcp
                   connection from your remote
                   host to this local host)
         - your remote hostname
         - your remote username
                 (which are used to start the
                  slave on the remote host using
                  the remote shell utility rsh)
         - and the shell used on the remote host
                 (see /etc/passwd there
                 possible values are csh and sh)
Example: rusc2 my_workstation my_username_on_my_workstation csh
```

Daraufhin kann die Zeile

```
local_hostname remote_hostname remote_username remote_loginshell
```

mit folgender beispielhafter Kombination eingegeben werden:

```
local_hostname    = rusc2
remote_hostname   = awssg5
remote_username   = Username auf awssg5
remote_loginshell = csh (wie in /etc/passwd eingetragen !)
```

Das rsh-Kommando zum Start des Server-Programms wird dann vom Client-Programm ua1c_cb automatisch abgesetzt.

Anmerkung:

Auf manchen Superrechnern, bei denen mit rsh eine Reduced Shell für Benutzer mit überzogenem Plattenkontingent gemeint ist, muß durch geeignete Suchpfade und Symbolic Links sichergestellt werden, daß tatsächlich ein Programm mit der Funktionalität einer Remote Shell aufgerufen wird.

15 Interner Aufbau der Stubs im Rel. 1.0

15.1 Die Stubs und ihre Komponenten

15.1.1 Aufbau des Stub-Presourcefiles

Der Stub-Generator übersetzt den RPI-File ziemlich direkt in die Stub-Presourcefiles. Am Beispiel $DFNRPCEXAMPLES/ua1a soll der Aufbau der Stubs erläutert werden. Die ausführliche Sprachdefinition findet man im Kapitel 15.5 auf Seite 283.

ACHTUNG: Die Groß- und Kleinschreibung ist signifikant

15.1.1.1 Declaration Part

ua1.rpi

```
C*[OPTION] SEND_BUFFER_LENGTH      =   2400
C*[OPTION] RECEIVE_BUFFER_LENGTH   =   2400
C*[OPTION] WORD_BUFFER_LENGTH      = 100000
C*[OPTION] CHARACTER_BUFFER_LENGTH =  10000
C*[OPTION] TRACE_DEPTH             =     20
C*[OPTION] OPERATION_BITS          =     10
C*[OPTION] OBJECTS_MAXIMUM         =    100
C*[OPTION] WAIT_TIME_NEXT_CALL     =   1000
C*[OPTION] WAIT_TIME_SEND          =    300
C*[OPTION] WAIT_TIME_CONNECT       =    100
C*[OPTION] WAIT_TIME_ACCEPT        =   2000
C*[OPTION] WAIT_TIME_RELEASE       =    200
C*[OPTION] WAIT_TIME_LOCAL         =    100
C*[BLK_OPT] MASTER_STATIC_AUTHORIZATION = 0
C*[BLK_OPT] SLAVE_STATIC_AUTHORIZATION  = 0
C*[MOD_PAR] sm ,DEPTH=0, BACKDEPTH=0 ,CODE=4
```

wird übersetzt in folgende Macros (in ua1a_cb_stb.m4 und ua1a_sb_stb.m4)

```
define({S1B_MAXIMUM},        {2400})
define({R1B_MAXIMUM},        {2400})
define({W1B_MAXIMUM},        {100000})
define({CB_MAXIMUM},         {10000})
define({TR_MAXIMUM_LEVEL},   {20})
define({OPERATION_BITS},     {10})
define({OBJECTS_MAXIMUM},    {100})
define({Wait_Time_Next_Call},{1000})
define({Wait_Time_Send},     {300})
define({Wait_Time_Connect},  {100})
define({Wait_Time_Accept},   {2000})
define({Wait_Time_Release},  {200})
define({Wait_Time_Local},    {100})
define({MASTER_STATIC_AUTHORIZATION},{0})
define({SLAVE_STATIC_AUTHORIZATION}, {0})
```

Die Module-Option DEPTH und CODE werden beim Umsetzen von C*[MOD_USE] benützt, DEPTH außerdem bei C*[MOD_BEG].

Bei weglassen der Macros wird der entsprechende Default angewandt.

15.1.1.2 Block Definition Part

ua1a.rpi

```
C*[BLK_BEG] cb, NAME='ua1 client block', DEPTH=0, CLIENT_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     cp, LOCAL
C*[MOD_USE]     sm, CLIENT
C*[BLK_END]
```

wird beim CLIENT übersetzt in (nur ua1a_cb_stb.m4)

```
BLOCK_BEGIN(CLIENT_SITE,cb,ua1 client block,0)
C 1. argument: CLIENT_SITE or SERVER_SITE
C 2. argument: short object name
C 3. argument: long  object name
C 4. argument: block depth

MODULE_USAGE(sm,4,CLIENT,0)
C 1. argument: short module name
C 2. argument: module code (must be identical on client and server site)
C 3. argument: module type
C 4. argument: module depth   (used only in SERVER modules)

BLOCK_END
```

und ua1a.rpi

```
C*[BLK_BEG] sb, NAME='ua1 server block', DEPTH=0, SERVER_SITE
C*[MOD_USE]     sp, LOCAL
C*[MOD_USE]     sm, SERVER
C*[BLK_END]
```

wird beim SERVER übersetzt in (nur ua1a_sb_stb.m4)

```
BLOCK_BEGIN(SERVER_SITE,sb,ua1 server block,0)
MODULE_USAGE(sm,4,SERVER,0)
BLOCK_END
```

und ua1a.rpi

```
C*[BLK_BEG] lb, NAME='ua1 local block', DEPTH=0, LOCAL_SITE
C*[MOD_USE]     cm, LOCAL
C*[MOD_USE]     sm, LOCAL
C*[BLK_END]
```

Hierzu entsteht natürlich kein Stub, da Stubs nur zu CLIENT_SITE und SERVER_SITE Blöcken generiert werden.

C*[MOD_USE] Statements werden nur bei module_type CLIENT oder SERVER in MODULE_USAGE Macros umgewandelt.

15.1.1.3 Module Definition Part

ua1.rpi

```
C*[MOD_BEG] sm, NAME='ua1 server routines'
```

wird beim CLIENT übersetzt in (nur ua1a_cb_stb.m4)

```
MODULE_BEGIN(CLIENT,sm,ua1 server routines,0,0)
C 1. argument: CLIENT or SERVER
C 2. argument: short module name
C 3. argument: long  module name
C 4. argument: module depth
C 5. argument: backdepth
```

und beim SERVER in (nur ua1a_sb_stb.m4)

```
MODULE_BEGIN(SERVER,sm,ua1 server routines,0,0)
```

und ua1.rpi

```
C*[RPI_BEG] ASYNCHRONOUS
      subroutine ropen1 (lun,filestring,statusstring)

      character*(*) filestring
C*RPC+                          [IN]
      character*(*) statusstring
C*RPC+                          [IN]
      integer lun
C*RPC+                          [IN]
C*[RPI_NON]
      open(lun,file=filestring,status=statusstring)
      return
      end
C*[RPI_END]
```

wird beim CLIENT übersetzt in (nur ua1a_cb_stb.m4)

```
CLIENT_STUB(ropen1,1,RPC_Asynchron_Call)
C 1. argument: subroutine name
C 2. argument: operation code
C 3. argument: RPC_Synchron_Call or RPC_Asynchron_Call
      SUBROUTINE ropen1(ARG01,ARG02,ARG03)
      INTEGER ARG01
      CHARACTER*(*) ARG02
      CHARACTER*(*) ARG03
      INTEGER DI0102,DJ0102,II0102,IJ0102
      INTEGER DI0103,DJ0103,II0103,IJ0103
      CLIENT_STUB_BEGIN
      DI0102=1
      DJ0102=LEN(ARG02)
      II0102=DI0102
      IJ0102=DJ0102
      DI0103=1
      DJ0103=LEN(ARG03)
      II0103=DI0103
      IJ0103=DJ0103
      MARSHALL_FIX_PART(40)
      MARSHALL_HEADER
C Part 1  - allocation length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0103)
C Part 2  - transfer length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0103)
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING(ARG02,II0102,(IJ0102-II0102+1))
      MARSHALL_STRING(ARG03,II0103,(IJ0103-II0103+1))
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_SEND_CALL
      MARSHALL_END

*** NO OF RETURN PROTOCOL UNIT
C

      CLIENT_STUB_END
```

(in asynchronous routines there isn't any unmarshalling part in the client stub)

```
      RETURN
      END
```

und beim SERVER in (nur ua1a_sb_stb.m4)

```
SERVER_STUB(ropen1,1,RPC_Asynchron_Call)
      SERVER_INTERFACE
      SERVER_STUB_BEGIN
      INTEGER ARG01
      INTEGER PST02
      INTEGER PST03
      INTEGER DI0102,DJ0102,II0102,IJ0102
      INTEGER LAL02
      INTEGER DI0103,DJ0103,II0103,IJ0103
      INTEGER LAL03
      UNMARSHALL_FIX_PART(40)
      UNMARSHALL_INVOCATION
C Part 1  - allocation length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0103)
      LAL02 = 1
     +        *(DJ0102 - DI0102 + 1)
      LAL03 = 1
     +        *(DJ0103 - DI0103 + 1)
      PST02=CB_Used+1
      CB_Used=CB_Used+LAL02
      PST03=CB_Used+1
      CB_Used=CB_Used+LAL03
C Part 2  - transfer length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0103)
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      UNMARSHALL_STRING(CBuffer,(PST02-DI0102+II0102),(IJ0102-II0102+1))
      UNMARSHALL_STRING(CBuffer,(PST03-DI0103+II0103),(IJ0103-II0103+1))
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CALL ropen1(
```

```
     +   ARG01,
     +   CBuffer(PST02:PST02+LAL02-1),
     +   CBuffer(PST03:PST03+LAL03-1))

*** NO OF RETURN PROTOCOL UNIT
C
```

(in asynchronous routines there isn't any marshalling part in the server stub)

```
      SERVER_STUB_END
      RETURN
      END
```

Hierbei sind folgende Regeln zu beachten:

1. Jedes Marshalling muß im richtigen *Part* eingeordnet sein.
2. Jedem Marshalling beim Client muß ein entsprechendes Unmarshalling beim Server entsprechen und umgekehrt.
3. Numerische Array-Argumente werden beim Server im WBuffer, Strings im CBuffer allokiert, indem ein Pointer *PAR...* berechnet wird und der entsprechende Zähler WB_Used, bzw. CB_Used hochgezählt wird.
4. In MARSHALL_FIX_PART, bzw. UNMARSHALL_FIX_PART muß als Argument die Anzahl der zu übertragenden Bytes aus den Parts 1 bis 3, zuzüglich 4 Bytes für den Header, eingetragen werden.
5. Alle Argumente müssen als INTEGER, INTEGER-Feld, STRING oder STRING-Feld deklariert werden.

ua1.rpi

```
C*[RPI_BEG]
      subroutine rwrite1 (lun,formatstr,iwrite,satzlaenge)
      character*10 formatstr
C*RPC+                           [IN]
      integer satzlaenge
C*RPC+                           [IN]
      integer lun
C*RPC+                           [IN,OUT]
      integer iwrite(satzlaenge)
C*RPC+                           [IN]
C.....alternative, falls die satzlaenge auch 0 sein darf:
C     integer iwrite(*)
C RPC+                           [IN, MAXINDEX=satzlaenge]
C*[RPI_NON]
      write(lun,formatstr) (iwrite(i),i=1,satzlaenge)
      write(*,'(10(i5))')  (iwrite(i),i=1,satzlaenge)
      return
      end
C*[RPI_END]
```

wird beim CLIENT übersetzt in (nur ua1a_cb_stb.m4)

```
CLIENT_STUB(rwrite1,2,RPC_Synchron_Call)
      SUBROUTINE rwrite1(ARG01,ARG02,ARG03,ARG04,ARG05)
      INTEGER ARG01
```

```
      CHARACTER*(*) ARG02
      INTEGER ARG03(*)
      INTEGER ARG04(*)
      INTEGER ARG05
      INTEGER DI0102,DJ0102,II0102,IJ0102
      INTEGER DI0103,DJ0103,II0103,IJ0103
      INTEGER DI0104,DJ0104,II0104,IJ0104
      CLIENT_STUB_BEGIN
      DI0102=1
      DJ0102=LEN(ARG02)
      II0102=0
      IJ0102=DJ0102-DI0102
      DI0103=1
      DJ0103=ARG05
      II0103=0
      IJ0103=DJ0103-DI0103
      DI0104=1
      DJ0104=ARG05
      II0104=0
      IJ0104=DJ0104-DI0104
      MARSHALL_FIX_PART(60)
      MARSHALL_HEADER
C Part 1  - allocation length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0104)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0104)
C Part 2  - transfer length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0102)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0103)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0104)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0104)
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG05)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      MARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,ARG03,1,II0103,(IJ0103-II0103+1))
      MARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,ARG04,1,II0104,(IJ0104-II0104+1))
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING(ARG02,(1+II0102),(IJ0102-II0102+1))
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
```

```
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_SEND_CALL
      MARSHALL_END
      UNMARSHALL_FIX_PART(8)
      UNMARSHALL_HEADER
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CLIENT_STUB_END
      RETURN
      END
```

und beim SERVER in (nur ua1a_sb_stb.m4)

```
SERVER_STUB(rwrite1,2,RPC_Synchron_Call)
      SERVER_INTERFACE
      SERVER_STUB_BEGIN
      INTEGER ARG01
      INTEGER PST02
      INTEGER PAR03
      INTEGER PAR04
      INTEGER ARG05
      INTEGER DI0102,DJ0102,II0102,IJ0102
      INTEGER LAL02
      INTEGER DI0103,DJ0103,II0103,IJ0103
      INTEGER LAL03
      INTEGER DI0104,DJ0104,II0104,IJ0104
      INTEGER LAL04
      UNMARSHALL_FIX_PART(60)
      UNMARSHALL_INVOCATION
C Part 1  - allocation length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0104)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0104)
      LAL02 = 1
     +        *(DJ0102 - DI0102 + 1)
```

```
      LAL03 = 1
     +        *(DJ0103 - DI0103 + 1)
      LAL04 = 1
     +        *(DJ0104 - DI0104 + 1)
      PST02=CB_Used+1
      CB_Used=CB_Used+LAL02
      PAR03=WB_Used+1
      WB_Used=WB_Used+LAL03
      PAR04=WB_Used+1
      WB_Used=WB_Used+LAL04
C Part 2  - transfer length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0102)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0103)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0104)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0104)
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG05)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      UNMARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,WBuffer,PAR03,II0103,(IJ0103-II0103+1))
      UNMARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,WBuffer,PAR04,II0104,(IJ0104-II0104+1))
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      UNMARSHALL_STRING(CBuffer,(PST02+II0102),(IJ0102-II0102+1))
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CALL rwrite1(
     +   ARG01,
     +   CBuffer(PST02:PST02+DJ0102-DI0102+1),
     +   WBuffer(PAR03),
     +   WBuffer(PAR04),
     +   ARG05)
      MARSHALL_FIX_PART(8)
      MARSHALL_RETURN
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
```

```
C Part 9  - strings (1 transfer byte)
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_SEND_RETURN
      MARSHALL_END
      SERVER_STUB_END
      RETURN
      END
```

<u>ua1.rpi</u>

```
C*[RPI_BEG] ASYNCHRONOUS BUFFERED
      subroutine rclose1 (lun)
      integer lun
C*RPC+              [IN]
C*[RPI_NON]
      close(lun)
      return
      end
C*[RPI_END]
```

<u>wird beim CLIENT übersetzt in</u> (nur ua1a_cb_stb.m4)

```
CLIENT_STUB(rclose1,3,RPC_Asynchron_Call)
      SUBROUTINE rclose1(ARG01)
      INTEGER ARG01
      CLIENT_STUB_BEGIN
      MARSHALL_FIX_PART(8)
      MARSHALL_HEADER
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_BUFFERED_CALL
      MARSHALL_END
      CLIENT_STUB_END
```

```
      RETURN
      END
```

und beim SERVER in (nur ua1a_sb_stb.m4)

```
SERVER_STUB(rclose1,3,RPC_Asynchron_Call)
      SERVER_INTERFACE
      SERVER_STUB_BEGIN
      INTEGER ARG01
      UNMARSHALL_FIX_PART(8)
      UNMARSHALL_INVOCATION
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG01)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CALL rclose1(
     +   ARG01)
      SERVER_STUB_END
      RETURN
      END
```

ua1.rpi

```
C*[MOD_END]
```

wird übersetzt in (ua1a_cb_stb.m4 und ua1a_sb_stb.m4)

```
MODULE_END
```

15.1.2 Weitere Beispiele

Die im Kapitel 5.7 auf S. 90 aufgeführten Beispiele findet man in $DFNRPCEXAMPLES/exa/exa.rpi und in den zugehörigen Stub-Presourcefiles $DFNRPCEXAMPLES/exa/exa_sm_clt.m4 und $DFNRPCEXAMPLES/exa/exa_sm_srv.m4.

Zwei Beispiele mit den wesentlichen Datentypen sind im folgenden abgedruckt:

sub10.rpi

```
C*[MOD_BEG] sm, NAME='examples server module'

C*[RPI_BEG]
C     Summary of types
C     ----------------

      SUBROUTINE sub10 (b, l,l1,l2,l4,l8, i,i1,i2,i4,i8,
     +                  r,r4,r8,r16, d, c,c8,c16,c32,s)

C            internal bytes on    SUN VAX CRAY2  | transfer
C            with compiler        f77 for cft77  | default
      BYTE             b
      LOGICAL          l
      LOGICAL*1        l1
      LOGICAL*2        l2
      LOGICAL*4        l4
C     LOGICAL*8        l8      only on hosts with 8 byte words
      INTEGER          i
      INTEGER*1        i1
      INTEGER*2        i2
      INTEGER*4        i4
C     INTEGER*8        i8      only on hosts with 8 byte words
      REAL             r
      REAL   *4        r4
      REAL   *8        r8
      REAL   *16       r16
      DOUBLE PRECISION d
      COMPLEX          c
      COMPLEX*8        c8
      COMPLEX*16       c16
      COMPLEX*32       c32
      CHARACTER*55     s

C*[RPI_NON]
      RETURN
      END
C*[RPI_END]

c*[MOD_END]
```

sub10_sm_clt.m4 (Clientstub-Presource)

```
MODULE_BEGIN(CLIENT,sm,examples server module,0,0)
CLIENT_STUB(sub10,1,RPC_Synchron_Call)
      SUBROUTINE sub10(ARG01,ARG02,ARG03,ARG04,ARG05,ARG06,ARG07
     + ,ARG08,ARG09,ARG10,ARG11,ARG12,ARG13,ARG14,ARG15,ARG16,ARG17
     + ,ARG18,ARG19,ARG20,ARG21)
      INT1 ARG01
      INTEGER ARG02
      INT1 ARG03
      INT2 ARG04
      INT4 ARG05
      INTEGER ARG06
      INTEGER ARG07
```

```
      INT1 ARG08
      INT2 ARG09
      INT4 ARG10
      INTEGER ARG11
      INTEGER ARG12
      INTEGER ARG13
      INTEGER ARG14(eval(8/Y_DEF_BYTES))
      INTEGER ARG15(eval(16/Y_DEF_BYTES))
      INTEGER ARG16(2)
      INTEGER ARG17(2)
      INTEGER ARG18(eval(8/Y_DEF_BYTES))
      INTEGER ARG19(eval(16/Y_DEF_BYTES))
      INTEGER ARG20(eval(32/Y_DEF_BYTES))
      CHARACTER*(*) ARG21
      INTEGER DI0121,DJ0121,II0121,IJ0121,OI0121,OJ0121
      CLIENT_STUB_BEGIN
      DI0121=1
      DJ0121=55
      II0121=0
      IJ0121=DJ0121-DI0121
      MARSHALL_FIX_PART(68)
      MARSHALL_HEADER
C Part 1  - allocation length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0121)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0121)
C Part 2  - transfer length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,II0121)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0121)
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(L,Y_DEF_BYTES,4,ARG02)
      MARSHALL_ITEM(L,4,4,ARG05)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG07)
      MARSHALL_ITEM(I,4,4,ARG10)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG12)
      MARSHALL_ITEM(R,4,4,ARG13)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,1)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,2)
      MARSHALL_ITEM(R,4,4,ARG18,1)
      MARSHALL_ITEM(R,4,4,ARG18,2)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
      MARSHALL_ITEM(L,2,2,ARG04)
      MARSHALL_ITEM(I,2,2,ARG09)
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
      MARSHALL_ITEM(I,1,1,ARG01)
      MARSHALL_ITEM(L,1,1,ARG03)
      MARSHALL_ITEM(I,1,1,ARG08)
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING(ARG21,(1+II0121),(IJ0121-II0121+1))
```

```
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
      MARSHALL_ITEM(R,8,8,ARG14,1)
      MARSHALL_ITEM(R,eval(2*Y_DEF_BYTES),8,ARG16,1)
      MARSHALL_ITEM(R,8,8,ARG19,1)
      MARSHALL_ITEM(R,8,8,ARG19,2)
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
      MARSHALL_ITEM(R,16,16,ARG15,1)
      MARSHALL_ITEM(R,16,16,ARG20,1)
      MARSHALL_ITEM(R,16,16,ARG20,2)
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_SEND_CALL
      MARSHALL_END
      UNMARSHALL_FIX_PART(60)
      UNMARSHALL_HEADER
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,OI0121)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,OJ0121)
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(L,Y_DEF_BYTES,4,ARG02)
      UNMARSHALL_ITEM(L,4,4,ARG05)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG07)
      UNMARSHALL_ITEM(I,4,4,ARG10)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG12)
      UNMARSHALL_ITEM(R,4,4,ARG13)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,1)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,2)
      UNMARSHALL_ITEM(R,4,4,ARG18,1)
      UNMARSHALL_ITEM(R,4,4,ARG18,2)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
      UNMARSHALL_ITEM(L,2,2,ARG04)
      UNMARSHALL_ITEM(I,2,2,ARG09)
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
      UNMARSHALL_ITEM(I,1,1,ARG01)
      UNMARSHALL_ITEM(L,1,1,ARG03)
      UNMARSHALL_ITEM(I,1,1,ARG08)
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      UNMARSHALL_STRING(ARG21,(1+OI0121),(OJ0121-OI0121+1))
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
      UNMARSHALL_ITEM(R,8,8,ARG14,1)
      UNMARSHALL_ITEM(R,eval(2*Y_DEF_BYTES),8,ARG16,1)
      UNMARSHALL_ITEM(R,8,8,ARG19,1)
      UNMARSHALL_ITEM(R,8,8,ARG19,2)
```

```
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
      UNMARSHALL_ITEM(R,16,16,ARG15,1)
      UNMARSHALL_ITEM(R,16,16,ARG20,1)
      UNMARSHALL_ITEM(R,16,16,ARG20,2)
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CLIENT_STUB_END
      RETURN
      END
MODULE_END
```

sub10_sm_srv.m4 (Serverstub-Presource)

```
MODULE_BEGIN(SERVER,sm,examples server module,0,0)
SERVER_STUB(sub10,1,RPC_Synchron_Call)
      SERVER_INTERFACE
      SERVER_STUB_BEGIN
      INT1 ARG01
      INTEGER ARG02
      INT1 ARG03
      INT2 ARG04
      INT4 ARG05
      INTEGER ARG06
      INTEGER ARG07
      INT1 ARG08
      INT2 ARG09
      INT4 ARG10
      INTEGER ARG11
      INTEGER ARG12
      INTEGER ARG13
      INTEGER ARG14(eval(8/Y_DEF_BYTES))
      INTEGER ARG15(eval(16/Y_DEF_BYTES))
      INTEGER ARG16(2)
      INTEGER ARG17(2)
      INTEGER ARG18(eval(8/Y_DEF_BYTES))
      INTEGER ARG19(eval(16/Y_DEF_BYTES))
      INTEGER ARG20(eval(32/Y_DEF_BYTES))
      INTEGER PST21
      INTEGER DI0121,DJ0121,II0121,IJ0121,OI0121,OJ0121
      INTEGER LAL21
      UNMARSHALL_FIX_PART(68)
      UNMARSHALL_INVOCATION
C Part 1  - allocation length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DI0121)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,DJ0121)
      LAL21 = 1
     +        *(DJ0121 - DI0121 + 1)
      PST21=CB_Used+1
      CB_Used=CB_Used+LAL21
C Part 2  - transfer length of arrays
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,II0121)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,IJ0121)
C Part 3  - items  with 4 transfer bytes
```

```
      UNMARSHALL_ITEM(L,Y_DEF_BYTES,4,ARG02)
      UNMARSHALL_ITEM(L,4,4,ARG05)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG07)
      UNMARSHALL_ITEM(I,4,4,ARG10)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG12)
      UNMARSHALL_ITEM(R,4,4,ARG13)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,1)
      UNMARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,2)
      UNMARSHALL_ITEM(R,4,4,ARG18,1)
      UNMARSHALL_ITEM(R,4,4,ARG18,2)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
      UNMARSHALL_ITEM(L,2,2,ARG04)
      UNMARSHALL_ITEM(I,2,2,ARG09)
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
      UNMARSHALL_ITEM(I,1,1,ARG01)
      UNMARSHALL_ITEM(L,1,1,ARG03)
      UNMARSHALL_ITEM(I,1,1,ARG08)
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      UNMARSHALL_STRING(CBuffer,(PST21+II0121),(IJ0121-II0121+1))
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
      UNMARSHALL_ITEM(R,8,8,ARG14,1)
      UNMARSHALL_ITEM(R,eval(2*Y_DEF_BYTES),8,ARG16,1)
      UNMARSHALL_ITEM(R,8,8,ARG19,1)
      UNMARSHALL_ITEM(R,8,8,ARG19,2)
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
      UNMARSHALL_ITEM(R,16,16,ARG15,1)
      UNMARSHALL_ITEM(R,16,16,ARG20,1)
      UNMARSHALL_ITEM(R,16,16,ARG20,2)
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_END
      CALL sub10(
     +   ARG01,
     +   ARG02,
     +   ARG03,
     +   ARG04,
     +   ARG05,
     +   ARG06,
     +   ARG07,
     +   ARG08,
     +   ARG09,
     +   ARG10,
     +   ARG11,
     +   ARG12,
     +   ARG13,
     +   ARG14,
```

```
     +   ARG15,
     +   ARG16,
     +   ARG17,
     +   ARG18,
     +   ARG19,
     +   ARG20,
     +   CBuffer(PST21:PST21+DJ0121-DI0121+1))
      OI0121=0
      OJ0121=DJ0121-DI0121
      MARSHALL_FIX_PART(60)
      MARSHALL_RETURN
C Part 2  - transfer length of arrays
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,OI0121)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,OJ0121)
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(L,Y_DEF_BYTES,4,ARG02)
      MARSHALL_ITEM(L,4,4,ARG05)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG07)
      MARSHALL_ITEM(I,4,4,ARG10)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG12)
      MARSHALL_ITEM(R,4,4,ARG13)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,1)
      MARSHALL_ITEM(R,Y_DEF_BYTES,4,ARG17,2)
      MARSHALL_ITEM(R,4,4,ARG18,1)
      MARSHALL_ITEM(R,4,4,ARG18,2)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
C Part 5  - items  with 2 transfer bytes
      MARSHALL_ITEM(L,2,2,ARG04)
      MARSHALL_ITEM(I,2,2,ARG09)
C Part 6  - arrays with 2 transfer bytes
C Part 7  - items  with 1 transfer byte
      MARSHALL_ITEM(I,1,1,ARG01)
      MARSHALL_ITEM(L,1,1,ARG03)
      MARSHALL_ITEM(I,1,1,ARG08)
C Part 8  - arrays with 1 transfer byte
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING(CBuffer,(PST21+OI0121),(OJ0121-OI0121+1))
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
      MARSHALL_ITEM(R,8,8,ARG14,1)
      MARSHALL_ITEM(R,eval(2*Y_DEF_BYTES),8,ARG16,1)
      MARSHALL_ITEM(R,8,8,ARG19,1)
      MARSHALL_ITEM(R,8,8,ARG19,2)
C Part 12 - arrays with 8 transfer bytes
C Part 13 - items  with 16 transfer bytes
      MARSHALL_ITEM(R,16,16,ARG15,1)
      MARSHALL_ITEM(R,16,16,ARG20,1)
      MARSHALL_ITEM(R,16,16,ARG20,2)
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_SEND_RETURN
```

```
      MARSHALL_END
      SERVER_STUB_END
      RETURN
      END
MODULE_END
```

sub11.rpi

```
C*[MOD_BEG] sm, NAME='examples server module'

C*[RPI_BEG]
C     Summary of arrays
C     -----------------

      SUBROUTINE sub11 (b, l,l1,l2,l4,l8, i,i1,i2,i4,i8,
     +                 r,r4,r8,r16, d, c,c8,c16,c32,s)

C            internal bytes on    SUN VAX CRAY2  | transfer
C            with compiler        f77 for cft77  | default
      BYTE            b  (77)
      LOGICAL         l  (77)
      LOGICAL*1       l1 (77)
      LOGICAL*2       l2 (77)
      LOGICAL*4       l4 (77)
C     LOGICAL*8       l8 (77)  only on hosts with 8 byte words
      INTEGER         i  (77)
      INTEGER*1       i1 (77)
      INTEGER*2       i2 (77)
      INTEGER*4       i4 (77)
C     INTEGER*8       i8 (77)  only on hosts with 8 byte words
      REAL            r  (77)
      REAL   *4       r4 (77)
      REAL   *8       r8 (77)
      REAL   *16      r16(77)
      DOUBLE PRECISION d  (77)
      COMPLEX         c  (77)
      COMPLEX*8       c8 (77)
      COMPLEX*16      c16(77)
      COMPLEX*32      c32(77)
      CHARACTER*55    s  (77)

C                                w = the compiler issues a warning

C*[RPI_NON]
      RETURN
      END
C*[RPI_END]

c*[MOD_END]
```

sub11_sm_clt.m4 (Clientstub-Presource)
Gekürzte Version

```
MODULE_BEGIN(CLIENT,sm,examples server module,0,0)
CLIENT_STUB(sub11,1,RPC_Synchron_Call)
      SUBROUTINE sub11(ARG01,ARG02,ARG03,ARG04,ARG05,ARG06,ARG07
     + ,ARG08,ARG09,ARG10,ARG11,ARG12,ARG13,ARG14,ARG15,ARG16,ARG17
     + ,ARG18,ARG19,ARG20,ARG21)
      INT1 ARG01(*)
      INTEGER ARG02(*)
      INT1 ARG03(*)
      INT2 ARG04(*)
      INT4 ARG05(*)
      INTEGER ARG06
      INTEGER ARG07(*)
      INT1 ARG08(*)
      INT2 ARG09(*)
      INT4 ARG10(*)
      INTEGER ARG11
      INTEGER ARG12(*)
      INTEGER ARG13(*)
      INTEGER ARG14(*)
      INTEGER ARG15(*)
      INTEGER ARG16(*)
      INTEGER ARG17(*)
      INTEGER ARG18(*)
      INTEGER ARG19(*)
      INTEGER ARG20(*)
      CHARACTER*(*) ARG21(*)
      CLIENT_STUB_BEGIN
      MARSHALL_FIX_PART(12)
      MARSHALL_HEADER
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      MARSHALL_ARRAYN(1,L,Y_DEF_BYTES,4,ARG02,1,0,77)
      MARSHALL_ARRAYN(1,L,4,4,ARG05,1,0,77)
      MARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,ARG07,1,0,77)
      MARSHALL_ARRAYN(1,I,4,4,ARG10,1,0,77)
      MARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,ARG12,1,0,77)
      MARSHALL_ARRAYN(1,R,4,4,ARG13,1,0,77)
      MARSHALL_ARRAYN_H(2,R,Y_DEF_BYTES,4,ARG17,1,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(Y_DEF_BYTES){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG17,1,0,2,2,0,77)
      MARSHALL_ARRAYN_H(2,R,4,4,ARG18,1,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(4){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG18,1,0,2,2,0,77)
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
      MARSHALL_ARRAYN(1,L,2,2,ARG04,1,0,77)
      MARSHALL_ARRAYN(1,I,2,2,ARG09,1,0,77)
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
```

```
      MARSHALL_ARRAYN(1,I,1,1,ARG01,1,0,77)
      MARSHALL_ARRAYN(1,L,1,1,ARG03,1,0,77)
      MARSHALL_ARRAYN(1,I,1,1,ARG08,1,0,77)
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING_ARR_H(2,ARG21,1,0,55,55,0,77)
      CALL RPMSA2(*LAB_ERR,ARG21(1:55),0,55,55,0,77)
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
      MARSHALL_ARRAYN(1,R,8,8,ARG14,1,0,77)
      MARSHALL_ARRAYN(1,R,eval(2*Y_DEF_BYTES),8,ARG16,1,0,77)
      MARSHALL_ARRAYN_H(2,R,8,8,ARG19,1,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(8){}C16G(8){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG19,1,0,2,2,0,77)
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_ARRAYN(1,R,16,16,ARG15,1,0,77)
      MARSHALL_ARRAYN_H(2,R,16,16,ARG20,1,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(16){}C16G(16){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG20,1,0,2,2,0,77)
      MARSHALL_SEND_CALL
      MARSHALL_END
      UNMARSHALL_FIX_PART(12)
      UNMARSHALL_HEADER
C Part 1  - allocation length of arrays
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      UNMARSHALL_ARRAYN(1,L,Y_DEF_BYTES,4,ARG02,1,0,77)
      UNMARSHALL_ARRAYN(1,L,4,4,ARG05,1,0,77)
      UNMARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,ARG07,1,0,77)
      UNMARSHALL_ARRAYN(1,I,4,4,ARG10,1,0,77)
      UNMARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,ARG12,1,0,77)
      UNMARSHALL_ARRAYN(1,R,4,4,ARG13,1,0,77)
      UNMARSHALL_ARRAYN_H(2,R,Y_DEF_BYTES,4,ARG17,1,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(Y_DEF_BYTES){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG17,1,0,2,2,0,77)
      UNMARSHALL_ARRAYN_H(2,R,4,4,ARG18,1,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(4){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG18,1,0,2,2,0,77)
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
      UNMARSHALL_ARRAYN(1,L,2,2,ARG04,1,0,77)
      UNMARSHALL_ARRAYN(1,I,2,2,ARG09,1,0,77)
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
      UNMARSHALL_ARRAYN(1,I,1,1,ARG01,1,0,77)
      UNMARSHALL_ARRAYN(1,L,1,1,ARG03,1,0,77)
      UNMARSHALL_ARRAYN(1,I,1,1,ARG08,1,0,77)
C Part 9  - strings (1 transfer byte)
```

```
      UN_MARSHALL_STRING_ARR_H(2,ARG21,1,0,55,55,0,77)
      CALL RPMSA2(*LAB_ERR,ARG21(1:55),0,55,55,0,77)
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
      UNMARSHALL_ARRAYN(1,R,8,8,ARG14,1,0,77)
      UNMARSHALL_ARRAYN(1,R,eval(2*Y_DEF_BYTES),8,ARG16,1,0,77)
      UNMARSHALL_ARRAYN_H(2,R,8,8,ARG19,1,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(8){}C16G(8){}TRANSTYPE_H(R),*LAB_ERR,
     +   ,ARG19,1,0,2,2,0,77)
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_ARRAYN(1,R,16,16,ARG15,1,0,77)
      UNMARSHALL_ARRAYN(2,R,16,16,ARG20,1,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(16){}C16G(16){}TRANSTYPE_H(R),*LAB_ERR,
     +   ARG20,1,0,2,2,0,77)
      UNMARSHALL_END
      CLIENT_STUB_END
      RETURN
      END


MODULE_END
```

sub11_sm_srv.m4 (Serverstub-Presource)
Gekürzte Version

```
MODULE_BEGIN(SERVER,sm,examples server module,0,0)
SERVER_STUB(sub11,1,RPC_Synchron_Call)
      SERVER_INTERFACE
      SERVER_STUB_BEGIN
      INTEGER PAR01,PAR02,PAR03,PAR04,PAR05,ARG06,PAR07,PAR08,PAR09
      INTEGER PAR10,ARG11,PAR12,PAR13,PAR14,PAR15,PAR16,PAR17,PAR18
      INTEGER PAR19,PAR20,PST21
      INTEGER LAL01,LAL02,LAL03,LAL04,LAL05,LAL07,LAL08,LAL09,LAL10
      INTEGER LAL12,LAL13,LAL14,LAL15,LAL16,LAL17,LAL18,LAL19,LAL20
      INTEGER LAL21
      UNMARSHALL_FIX_PART(12)
      UNMARSHALL_INVOCATION
C Part 1  - allocation length of arrays
      LAL01 = 77
      LAL02 = 77
      LAL03 = 77
      LAL04 = 77
      LAL05 = 77
      LAL07 = 77
      LAL08 = 77
      LAL09 = 77
      LAL10 = 77
      LAL12 = 77
      LAL13 = 77
      LAL14 = 77
      LAL15 = 77
```

```
LAL16 = 77
LAL17 = 77
LAL18 = 77
LAL19 = 77
LAL20 = 77
LAL21 = 55*77
ifelse(eval(Y_MIN_BYTES>1),1,{
  PAR01=WB_Used+1
  WB_Used=WB_Used+LAL01
},{
  PAR01=WB_Used*4+1
  WB_Used=WB_Used+(LAL01+3)/4
})
PAR02=WB_Used+1
WB_Used=WB_Used+LAL02
ifelse(eval(Y_MIN_BYTES>1),1,{
  PAR03=WB_Used+1
  WB_Used=WB_Used+LAL03
},{
  PAR03=WB_Used*4+1
  WB_Used=WB_Used+(LAL03+3)/4
})
ifelse(eval(Y_MIN_BYTES>2),1,{
  PAR04=WB_Used+1
  WB_Used=WB_Used+LAL04
},{
  PAR04=WB_Used*2+1
  WB_Used=WB_Used+(LAL04+1)/2
})
PAR05=WB_Used+1
WB_Used=WB_Used+LAL05
PAR07=WB_Used+1
WB_Used=WB_Used+LAL07
ifelse(eval(Y_MIN_BYTES>1),1,{
  PAR08=WB_Used+1
  WB_Used=WB_Used+LAL08
},{
  PAR08=WB_Used*4+1
  WB_Used=WB_Used+(LAL08+3)/4
})
ifelse(eval(Y_MIN_BYTES>2),1,{
  PAR09=WB_Used+1
  WB_Used=WB_Used+LAL09
},{
  PAR09=WB_Used*2+1
  WB_Used=WB_Used+(LAL09+1)/2
})
PAR10=WB_Used+1
WB_Used=WB_Used+LAL10
PAR12=WB_Used+1
WB_Used=WB_Used+LAL12
PAR13=WB_Used+1
WB_Used=WB_Used+LAL13
ifelse(Y_DEF_BYTES,4,{WB_Used=((WB_Used+1)/2)*2})
```

```
      PAR14=WB_Used ifelse(Y_DEF_BYTES,4,{/2})+1
      WB_Used=WB_Used+LAL14 ifelse(Y_DEF_BYTES,4,{*2})
      ifelse(Y_DEF_BYTES,4,{
         WB_Used=((WB_Used+3)/4)*4
      },{
         WB_Used=((WB_Used+1)/2)*2
      })
      PAR15=WB_Used/eval(16/Y_DEF_BYTES)+1
      WB_Used=WB_Used+LAL15*eval(16/Y_DEF_BYTES)
      WB_Used=((WB_Used+1)/2)*2
      PAR16=WB_Used/2+1
      WB_Used=WB_Used+LAL16*2
      PST21=CB_Used+1
      CB_Used=CB_Used+LAL21
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      UNMARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      UNMARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      UNMARSHALL_ARRAYN(1,L,Y_DEF_BYTES,4,WBuffer,PAR02,0,77)
      UNMARSHALL_ARRAYN(1,L,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR05,0,77)
      UNMARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,WBuffer,PAR07,0,77)
      UNMARSHALL_ARRAYN(1,I,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR10,0,77)
      UNMARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,WBuffer,PAR12,0,77)
      UNMARSHALL_ARRAYN(1,R,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR13,0,77)
      UNMARSHALL_ARRAYN_H(2,R,Y_DEF_BYTES,4,WBuffer,PAR17,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(Y_DEF_BYTES){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR17,0,2,2,0,77)
      UNMARSHALL_ARRAYN_H(2,R,4,4,WBuffer,PAR18,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(4){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR18,0,2,2,0,77)
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
      UNMARSHALL_ARRAYN(1,L,2,2,W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer,PAR04,0,77)
      UNMARSHALL_ARRAYN(1,I,2,2,W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer,PAR09,0,77)
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
      UNMARSHALL_ARRAYN(1,I,1,1,WBuffer,PAR01,0,77)
      UNMARSHALL_ARRAYN(1,L,1,1,W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer,PAR03,0,77)
      UNMARSHALL_ARRAYN(1,I,1,1,W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer,PAR08,0,77)
C Part 9  - strings (1 transfer byte)
      UN_MARSHALL_STRING_ARR_H(2,CBuffer,PST21,0,55,55,0,77)
      CALL RPMSA2(*LAB_ERR,CBuffer(PST21:PST21+54),0,55,55,0,77)
C Part 10 - string arrays (1 transfer byte)
      UNMARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
      UNMARSHALL_ARRAYN(1,R,8,8,WBuffer,PAR14,0,77)
      UNMARSHALL_ARRAYN(1,R,eval(2*Y_DEF_BYTES),8,WBuffer,PAR16,0,77)
      UNMARSHALL_ARRAYN_H(2,R,8,8,WBuffer,PAR19,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(8){}C16G(8){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR19,0,2,2,0,77)
C Part 13 - items  with 16 transfer bytes
```

```
C Part 14 - arrays with 16 transfer bytes
      UNMARSHALL_ARRAYN(1,R,16,16,WBuffer,PAR15,0,77)
      UNMARSHALL_ARRAYN(2,R,16,16,WBuffer,PAR20,0,2,2,0,77)
      CALL RPMUA2(RPU{}C16G(16){}C16G(16){}TRANSTYPE_H(R),*LAB_ERR,
     +    WBuffer,PAR20,0,2,2,0,77)
      UNMARSHALL_END
      CALL sub11(
     +   WBuffer(PAR01),
     +   WBuffer(PAR02),
     +   W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer(PAR03),
     +   W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer(PAR04),
     +   W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer(PAR05),
     +   ARG06,
     +   WBuffer(PAR07),
     +   W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer(PAR08),
     +   W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer(PAR09),
     +   W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer(PAR10),
     +   ARG11,
     +   WBuffer(PAR12),
     +   WBuffer(PAR13),
     +   WBuffer(PAR14 ifelse(Y_DEF_BYTES,4,{*2-1})),
     +   WBuffer(PAR15 ifelse(Y_DEF_BYTES,4,{*4-3},{*2-1})),
     +   WBuffer(PAR16*2-1),
     +   WBuffer(PAR17),
     +   WBuffer(PAR18),
     +   WBuffer(PAR19),
     +   WBuffer(PAR20),
     +   CBuffer(PST21:PST21+55-1+1))
      MARSHALL_FIX_PART(12)
      MARSHALL_RETURN
C Part 2  - transfer length of arrays
C Part 3  - items  with 4 transfer bytes
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG06)
      MARSHALL_ITEM(I,Y_DEF_BYTES,4,ARG11)
      MARSHALL_VAR_PART
C Part 4  - arrays with 4 transfer bytes
      MARSHALL_ARRAYN(1,L,Y_DEF_BYTES,4,WBuffer,PAR02,0,77)
      MARSHALL_ARRAYN(1,L,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR05,0,77)
      MARSHALL_ARRAYN(1,I,Y_DEF_BYTES,4,WBuffer,PAR07,0,77)
      MARSHALL_ARRAYN(1,I,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR10,0,77)
      MARSHALL_ARRAYN(1,R,Y_DEF_BYTES,4,WBuffer,PAR12,0,77)
      MARSHALL_ARRAYN(1,R,4,4,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer,PAR13,0,77)
      MARSHALL_ARRAYN_H(2,R,Y_DEF_BYTES,4,WBuffer,PAR17,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(Y_DEF_BYTES){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
      +   WBuffer,PAR17,0,2,2,OI0117,77)
      MARSHALL_ARRAYN_H(2,R,4,4,WBuffer,PAR18,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(4){}C16G(4){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR18,0,2,2,0,77)
C Part 5  - items  with 2 transfer bytes
C Part 6  - arrays with 2 transfer bytes
      MARSHALL_ARRAYN(1,L,2,2,W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer,PAR04,0,77)
      MARSHALL_ARRAYN(1,I,2,2,W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer,PAR09,0,77)
C Part 7  - items  with 1 transfer byte
C Part 8  - arrays with 1 transfer byte
```

```
      MARSHALL_ARRAYN(1,I,1,1,WBuffer,PAR01,0,77)
      MARSHALL_ARRAYN(1,L,1,1,W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer,PAR03,0,77)
      MARSHALL_ARRAYN(1,I,1,1,W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer,PAR08,0,77)
C Part 9  - strings (1 transfer byte)
      MARSHALL_STRING_ARR_H(2,CBuffer,PST21,0,55,55,0,77)
      CALL RPMSA2(*LAB_ERR,CBuffer(PST21:PST21+54),0,55,55,0,77)
C Part 10 - string arrays (1 transfer byte)
      MARSHALL_ALIGNMENT
C Part 11 - items  with 8 transfer bytes
C Part 12 - arrays with 8 transfer bytes
      MARSHALL_ARRAYN(1,R,8,8,WBuffer,PAR14,0,77)
      MARSHALL_ARRAYN(1,R,eval(2*Y_DEF_BYTES),8,WBuffer,PAR16,0,77)
      MARSHALL_ARRAYN_H(2,R,8,8,WBuffer,PAR19,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(8){}C16G(8){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR19,0,2,2,0,77)
C Part 13 - items  with 16 transfer bytes
C Part 14 - arrays with 16 transfer bytes
      MARSHALL_ARRAYN(1,R,16,16,WBuffer,PAR15,0,77)
      MARSHALL_ARRAYN_H(2,R,16,16,WBuffer,PAR20,0,2,2,0,77)
      CALL RPMUA2(RPM{}C16G(16){}C16G(16){}TRANSTYPE_H(R),*LAB_ERR,
     +   WBuffer,PAR20,0,2,2,0,77)
      MARSHALL_SEND_RETURN
      MARSHALL_END
      SERVER_STUB_END
      RETURN
      END

MODULE_END
```

15.1.3 Umwandeln des Stub-Presourcefiles

Mit **$DFNRPCBIN/dfnrpcm4** kann man einen Stub-Presourcefile in einen Stub-Sourcefile umwandeln:

```
$DFNRPCBIN/dfnrpcm4  [-T transfertype]
                     [-M manufacturer]
                     [-A architecture]
                     [-O operatingsystem]
                     [-F fortrancompiler]
                     presource_stub_file.m4
```

Der Name des Stub-Presourcefile muß die Endung **.m4** haben. Er wird in den Stubsourcefile mit der Endung **.f** umgewandelt.

Die Optionen sind in Kapitel 7 „Der Stub-Generator“ auf S. 159 beschrieben.

15.2 Datentypdeklarationen in den Stub-Presourcefiles (Single Items)

```
Statements in den Stub bei einzelnen Items als Dummy Arguments

Datentyp                 Server                             Client
---------------------------------------------------------------------------
INTEGER ARGx             INTEGER ARGx                       <===
INTEGER*1 ARGx           INT1 ARGx                          <===
INTEGER*2 ARGx           INT2 ARGx                          <===
INTEGER*4 ARGx           INT4 ARGx                          <===
INTEGER*8 ARGx           INT8 ARGx                          <===

LOGICAL ARGx             INTEGER ARGx                       <===
LOGICAL*1 ARGx           INT1 ARGx                          <===
LOGICAL*2 ARGx           INT2 ARGx                          <===
LOGICAL*4 ARGx           INT4 ARGx                          <===
LOGICAL*8 ARGx           INT8 ARGx                          <===

BYTE ARGx                INT1 ARGx                          <===

REAL ARGx                INTEGER ARGx                       <===
REAL*4 ARGx              INTEGER ARGx                       <===
REAL*8 ARGx              INTEGER ARGx(eval(8/Y_DEF_BYTES))  <===
REAL*16 ARGx             INTEGER ARGx(eval(16/Y_DEF_BYTES)) <===

DOUBLEPRECISION ARGx     INTEGER ARGx(2)                    <===

COMPLEX ARGx             INTEGER ARGx(2)                    <===
COMPLEX*8 ARGx           INTEGER ARGx(eval(8/Y_DEF_BYTES))  <===
COMPLEX*16 ARGx          INTEGER ARGx(eval(16/Y_DEF_BYTES)) <===
COMPLEX*32 ARGx          INTEGER ARGx(eval(32/Y_DEF_BYTES)) <===

CHARACTER*(*) ARGx       CHARACTER*(*) ARGx                 <===
```

15.3 Speicherallokierung und zusätzliche Statements für verschiedene Datentypen (Arrays)

```
Statements in den Stub bei Arrays

Beim Server wird fuer jedes Dummy Argument eine Variable
                                              INTEGER LALx
generiert. Diese Variable erhaelt den Wert
                                              LALx=1
                                                   *(DJyx-DIyx+1)
zugewiesen (y=1 ... Anzahl der Dimensionen von ARGx)



Datentypdeklaration und zusaetzliche Statements beim Server
---------------------------------------------------------------------------
      INTEGER ARGx(*) ebenso: LOGICAL

Client:
      INTEGER ARGx(*)

SERVER:
      INTEGER PARx

      INTEGER LALx

      PARx=WB_Used+1
      WB_Used=WB_Used+LALx

      CALL ...(...,WBuffer(PARx),...)
---------------------------------------------------------------------------
      INTEGER*1 ARGx(*) ebenso: LOGICAL*1, BYTE

Client:
      INT1 ARGx(*)

Server:
      INTEGER PARx

      INTEGER LALx

      ifelse(eval(Y_MIN_BYTES>1),1,{
        PARx=WB_Used+1
        WB_Used=WB_Used+LALx
      },{
        PARx=WB_Used*4+1
        WB_Used=WB_Used+(LALx+3)/4
      })

      CALL ...(...,W{}ifelse(eval(Y_MIN_BYTES>1),1,{},{1})Buffer(PARx),...)
---------------------------------------------------------------------------
      INTEGER*2 ARGx(*) ebenso: LOGICAL*2
```

```
Client:
      INT2 ARGx(*)

Server:
      INTEGER PARx

      ifelse(eval(Y_MIN_BYTES>1),1,{
        PARx=WB_Used+1
        WB_Used=WB_Used+LALx
      },{
        PARx=WB_Used*2+1
        WB_Used=WB_Used+(LALx+1)/2
      })

      CALL ...(...,W{}ifelse(eval(Y_MIN_BYTES>2),1,{},{2})Buffer(PARx),...)
-----------------------------------------------------------------------------
      INTEGER*4 ARGx(*) ebenso: LOGICAL*4

Client:
      INT4 ARGx(*)

Server:
      Integer PARx

      PARx=WB_Used+1
      WB_Used=WB_Used+LALx

      CALL ...(...,W{}ifelse(eval(Y_MIN_BYTES>4),1,{},{4})Buffer(PARx),...)
-----------------------------------------------------------------------------
      INTEGER*8 ARGx(*) ebenso: LOGICAL*8

Client:
      INT*8 ARGx(*)

Server:
      INTEGER PARx

      ifelse(Y_DEF_BYTES,4,{
        ERRMSG(x. Argument: INTEGER*8 not supported on hosts with 4 byte
        word length)
      },{
        PARx=WB_Used+1
        WB_Used=WB_Used+LALx
      })

      CALL ...(...,WBuffer(PARx),...)

-----------------------------------------------------------------------------
      REAL ARGx(*)

Client:
      INTEGER ARGx(*)

Server:
```

```
      INTEGER PARx

      PARx=WB_Used+1
      WB_Used=WB_Used+LALx

      CALL ...(...,WBuffer(PARx),...)
---------------------------------------------------------------------------
      REAL*4 ARGx(*)

Client:
      INTEGER ARGx(*)

Server:
      INTEGER PARx(*)

      PARx=WB_Used+1
      WB_Used=WB_Used+LALx

      CALL ...(...,WBuffer(PARx),...)
---------------------------------------------------------------------------
      REAL*8 ARGx(*)

Slient:
      INTEGER ARGx(*)

Server:
      INTEGER PARx

      ifelse(Y_DEF_BYTES,4,{WB_Used=((WB_Used+1)/2)*2})
        PARx=WB_Used ifelse(Y_DEF_BYTES,4,{/2})+1
        WB_Used=WB_Used+LALx ifelse(Y_DEF_BYTES,4,{*2})

      CALL ...(...,WBuffer(PARx ifelse(Y_DEF_BYTES,4,{*2-1})),...)
---------------------------------------------------------------------------
      REAL*16 ARGx(*)

Client:
      INTEGER ARGx(*)

Server:
      INTEGER PARx

      ifelse(Y_DEF_BYTES,4,{
        WB_Used=((WB_Used+3)/4)*4
      },{
        WB_Used=((WB_Used+1)/2)*2
      })
      PARx=WB_Used/eval(16/Y_DEF_BYTES)+1
      WB_Used=WB_Used+LALx*eval(16/Y_DEF_BYTES)

      CALL ...(...,WBuffer(PARx ifelse(Y_DEF_BYTES,4,{*4-3},{*2-1})),...)
---------------------------------------------------------------------------
      DOUBLEPRECISION ARGx(*)
```

```
Client:
      INTEGER ARgx(*)

Server:
      INTEGER PARx

      WB_Used=((WB_Used+1)/2)*2
      PARx=WB_Used/2+1
      WB_Used=WB_Used+LALx*2

      CALL ...(...,WBuffer(PARx),...)
--------------------------------------------------------------------------
```

15.4 Transfer Parts für verschiedene Datentypen

Die Zuordnung des Transfer-Parts richtet sich immer nach der Transfer-Länge des zu übertragenden Arguments. Diese wiederum richtet sich nach:
1. TRANSFER*x Angabe im Qualifier
2. nach Transfer-Default des jeweiligen Datentyps

Beim Datentyp Complex bezieht sich die Angabe TRANSFER*x immer auf die Länge einer Komponente, die Angabe COMPLEX*x jedoch auf die Gesamtlänge.

```
Part 3        [TRANSFER*4]       Single Argument
              INTEGER            Single Argument
              INTEGER*4          Single Argument
              LOGICAL            Singel Argument
              LOGICAL*4          Single Argument
              REAL               Single Argument
              REAL*4             Single Argument
              COMPLEX            Single Argument
              COMPLEX*8          Single Argument

Part 4        [TRANSFER*4]       Array
              INTEGER            Array
              INTEGER*4          Array
              LOGICAL            Array
              LOGICAL*4          Array
              REAL               Array
              REAL*4             Array

Part 5        [TRANSFER*2]       Single Argument
              INTEGER*2          Single Argument
              LOGICAL*2          Single Argument

Part 6        [TRANSFER*2]       Array
              INTEGER*2          Array
              LOGICAL*2          Array

Part 7        [TRANSFER*1]       Single Argument
```

```
              INTEGER*1          Single Argument
              LOGICAL*1          Single Argument
              BYTE               Single Argument

Part 8        [TRANSFER*1]       Array
              INTEGER*1          Array
              LOGICAL*1          Array
              BYTE               Array

Part 9        CHARACTER

Part 10       ALIGNMENT

Part 11       [TRANSFER*8]       Single Argument
              INTEGER*8          Single Argument
              REAL*8             Single Argument
              DOUBLEPRECISION    Single Argument
              COMPLEX*16         Single Argument

Part 12       [TRANSFER*8]       Array
              INTEGER*8          Array
              REAL*8             Array
              DOUBLEPRECISION    Array
              COMPLEX*16         Array

Part 13       [TRANSFER*16]      Single Argument
              REAL*16            Single Argument
              COMPLEX*32         Single Argument

Part 14       [TRANSFER*16]      Array
              REAL*16            Array
              COMPLEX*32         Array
```

15.5 Die Stub-Presource Sprachdefinition

Der Stub-Generator übersetzt den RPI-File in die Stub-Presourcefiles. Eine Einführung findet man im Kapitel 15.1.1 auf Seite 253.

15.5.1 Allgemeine Syntax

Ein Stub-Presourcefile besteht aus

- DFN-RPC Macros
- Regulären Fortran Statements
- Fortran Kommentarzeilen

Die DFN-RPC Macros mit deklarierendem Character beginnen in Spalte 1, während diejenigen, welche in echten Fortrancode umgewandelt werden, in den Spalten 7-9 (beliebig) beginnen müssen.

ACHTUNG: Die Groß- und Kleinschreibung ist signifikant

Bei den DFN-RPC Macro Aufrufen dürfen **keine Leerzeichen** eingestreut werden. (Ausnahmen sind nur im Declaration Part an der dort angegebenen Stelle erlaubt.)

15.5.2 Aufbau eines Stub-Presourcefiles

Bei gemeinsamer Übersetzung von Block- und Moduldefinitionen:

declaration part
block definition part
module definition part

Bei getrennter Übersetzung von Block- und Moduldefinitionen:

declaration part
block definition part

und

declaration part
module definition part

15.5.3 Declaration Part

Der *Declaration Part* kann folgende Zeilen enthalten

```
define({S1B_MAXIMUM},        {2400})
define({R1B_MAXIMUM},        {2400})
define({W1B_MAXIMUM},        {100000})
define({CB_MAXIMUM},         {10000})
define({TR_MAXIMUM_LEVEL},   {20})
define({OPERATION_BITS},     {10})
define({OBJECTS_MAXIMUM},    {100})
define({Wait_Time_Next_Call},{1000})
define({Wait_Time_Send},     {300})
define({Wait_Time_Connect},  {100})
define({Wait_Time_Accept},   {2000})
define({Wait_Time_Release},  {200})
define({Wait_Time_Local},    {100})
define({MASTER_STATIC_AUTHORIZATION},{0})
define({SLAVE_STATIC_AUTHORIZATION}, {0})
```

15.5.4 Block Definition Part

Der *Block Definition Part* besteht aus **genau einer**

block definition

Im RPI-File sind mehrere Block Definitions zulässig. Zu jeder dieser Block Definitions wird *ein* getrennter Stub-Presourcefile erzeugt.

15.5.5 Block Definition

Die *Block Definition* hat folgenden Aufbau:

```
BLOCK_BEGIN(___,___,___,___)
MODULE_USAGE(___,___,___,___)
  ...
MODULE_USAGE(___,___,___,___)
BLOCK_END
```

Die im RPI-File enthaltenen *LOCAL* Module entfallen in den Stub-Presourcefiles.

Beispiel:

```
BLOCK_BEGIN(CLIENT_SITE,cb,ua1 client block,0)
MODULE_USAGE(sm,4,CLIENT,0)
BLOCK_END
```

Syntax der Macros:

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          BLOCK_BEGIN                                    |
|                                                               |
|                                                               |
| Task           Begin of a block, which contains all stubs     |
|                of a client or a server site                   |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=SITE         (CLIENT_SITE or SERVER_SITE)                  |
| $2=S_OBJNAM     text             In short object name         |
| $3=L_OBJNAM     text             In long  object name         |
| $4=MAXLEV       integer_value >= -1  depth for generating      |
|                                      block dispatchers        |
|                 -1= does not generate a block dispatcher      |
|                  0= generates block dispatcher at level 1     |
|                     which is recursively callable             |
|                  1= generates block dispatcher at level 1     |
|                     which is NOT recursively callable         |
|                  2= generates block dispatcher at levels 1-2  |
|                  3= generates block dispatcher at levels 1-3  |
|                                                               |
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MODULE_USAGE                                   |
|                                                               |
|                                                               |
| Task           includes one module into a block               |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=S_MODNAM     text             In  short module name        |
| $2=MODCOD       integer_value >0 In  module code              |
| $3=MODTYP       (CLIENT,SERVER)  In  type of module           |
```

```
|                  SERVER means that this module contains at    |
|                  least one SERVER_STUB. (The Fortran Remote   |
|                  Procedure Interface Definition Language      |
|                  F-RPIDL does not support modules with 'calls'|
|                  and 'callbacks' mixed, that is with client   |
|                  and server stubs mixed in the .m4 input to   |
|                  this macro package. This macro package       |
|                  supports mixed modules.)                     |
| $4=MAXLEV       integer_value >= -1  depth for calling the   |
|                                      module dispatcher from  |
|                                      the block dispatcher    |
|                                      if MODTYP=SERVER        |
|                 -1= module dispatcher does not exist         |
|                     (nonsense if MODTYP=SERVER)              |
|                  0= module dispatcher exists at level 1      |
|                     and is recursively callable              |
|                  1= module dispatcher exists at level 1      |
|                     and is NOT recursively callable          |
|                  2= module dispatcher exists at levels 1 to 2|
|                  3= module dispatcher exists at levels 1 to 3|
|                                                               |
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          BLOCK_END                                      |
|                                                               |
|                                                               |
| Task           end of a block                                 |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------
```

15.5.6 Module Definition Part

Der *Module Definition Part* besteht aus einer oder mehreren

module definition

Die im RPI-File enthaltenen *LOCAL* Module entfallen in den Stub-Presourcefiles.

15.5.7 Module Definition

Eine *Module Definition* hat folgenden Aufbau:

```
MODULE_BEGIN(___,___,___,___,___)
stub definition of one procedure
  ...
stub definition of one procedure
MODULE_END
```

Routinen, die nicht mit C*[RPI BEG] ... C*[RPI END] im RPI-File geklammert sind, d.h. nicht zum Remote Procedure Interface des Moduls gehören, entfallen in den Stub-Presourcefiles.

Beispiel:

MODULE_BEGIN(CLIENT,sm,ua1 server routines,0,0)
client stub definition of ropen1
client stub definition of rwrite1
client stub definition of rclose1
MODULE_END

Syntax der Macros:

```
----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          MODULE_BEGIN                                  |
|                                                              |
|                                                              |
| Task           Begin of a module                             |
|                                                              |
| Parameter       Type            I/O Usage                    |
| $1=MODTYP       (CLIENT,SERVER) In  type of module           |
|                 SERVER means that this module contains at    |
|                 least one SERVER_STUB. (The Fortran Remote   |
|                 Procedure Interface Definition Language      |
|                 F-RPIDL does not support modules with 'calls'|
|                 and 'callbacks' mixed, that is with client   |
|                 and server stubs mixed in the .m4 input to   |
|                 this macro package. This macro package       |
|                 supports mixed modules.)                     |
| $2=S_MODNAM     text            In  short module name        |
| $3=L_MODNAM     text            In  long  module name        |
| $4=MAXLEV       integer_value >= -1  depth for generating    |
|                                      module dispatchers      |
|                                      if MODTYP=SERVER        |
|                 -1= does not generate a module dispatcher    |
|                  0= generates module dispatcher at level 1   |
|                     which is recursively callable            |
|                  1= generates module dispatcher at level 1   |
|                     which is NOT recursively callable        |
|                  2= generates module dispatcher at levels 1-2|
|                  3= generates module dispatcher at levels 1-3|
| $5=MAXBAC       integer_value >= -1  depth for calling the   |
|                                      block dispatcher from   |
|                                      each client stub        |
|                 -1= block dispatcher does not exist          |
|                  0= block dispatcher exists at level 1       |
|                     and is recursively callable              |
|                  1= block dispatcher exists at level 1       |
|                     and is NOT recursively callable          |
|                  2= block dispatcher exists at levels 1 to 2 |
|                  3= block dispatcher exists at levels 1 to 3 |
|                                                              |
----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MODULE_END                                     |
|                                                               |
|                                                               |
| Task           End of a module                                |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------
```

15.5.8 Stub Definitition einer Procedure

Eine *Stub Definition einer Procedure* besteht entweder aus einem

Client Stub einer Procedure

oder aus einem

Server Stub einer Procedure

Die im RPI-File enthaltenen *LOCAL* Module entfallen in den Stub-Presourcefiles.

15.5.9 Client- und Serverstub einer Procedure

Der prinzipiell Aufbau ist in den Beispielen in Kap. 15.1.2 auf S. 263 ersichtlich.

15.5.10 Client Stub Macros

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          CLIENT_STUB                                    |
|                                                               |
|                                                               |
| Task           Declaration of a client stub                   |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=OPNAME       text            I   rpc operation name        |
| $2=OPCODE       integer_value   I   rpc operation code        |
| $3=SYNC_MODE    (RPC_Synchron_Call, synchronisation mode      |
|                  RPC_Asynchron_Call)                          |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
```

```
| Macro          CLIENT_STUB_BEGIN                              |
|                                                               |
|                                                               |
| Task           Initialization of one client stub subroutine   |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (None)                                                        |
|                                                               |
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          CLIENT_STUB_END                                |
|                                                               |
|                                                               |
| Task           End of one client stub subroutine              |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (None)                                                        |
|                                                               |
-----------------------------------------------------------------
```

15.5.11 Server Stub Macros

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          SERVER_STUB                                    |
|                                                               |
|                                                               |
| Task           Declaration of a server stub                   |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=OPNAME       text            I   rpc operation name        |
| $2=OPCODE       integer_value   I   rpc operation code        |
| $3=SYNC_MODE    (RPC_Synchron_Call, synchronisation mode      |
|                  RPC_Asynchron_Call)                          |
|                                                               |
-----------------------------------------------------------------
```

```
-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          SERVER_INTERFACE                               |
|                                                               |
|                                                               |
| Task           Interface of a server stub                     |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
```

```
----------------------------------------------------------------

----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          SERVER_STUB_BEGIN                             |
|                                                              |
|                                                              |
| Task           Initialization of one server stub subroutine  |
|                                                              |
| Parameter       Type            I/O Usage                    |
| (None)                                                       |
|                                                              |
----------------------------------------------------------------

----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          SERVER_STUB_END                               |
|                                                              |
|                                                              |
| Task           End of one server stub subroutine             |
|                                                              |
| Parameter       Type            I/O Usage                    |
| (None)                                                       |
|                                                              |
----------------------------------------------------------------
```

15.5.12 Marshalling Macros

```
----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          MARSHALL_FIX_PART                             |
|                                                              |
|                                                              |
| Task           Initialization of the fixed part.             |
|                                                              |
| Parameter       Type            I/O Usage                    |
| $1=NUM_OF_BYTES integer_value   I   must be reserved in the  |
|                                     send buffer              |
|                                                              |
----------------------------------------------------------------

----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          MARSHALL_HEADER                               |
|                                                              |
|                                                              |
| Task           Compute and send the header item.             |
```

```
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MARSHALL_RETURN                                |
|                                                               |
|                                                               |
| Task           Compute and send the RETURN item.              |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MARSHALL_ITEM                                  |
|                                                               |
|                                                               |
| Task           Marshalling (convert and send) one dummy       |
|                argument                                       |
|                                                               |
| Parameter       Type            I/O Usage                     |
|                                                               |
| $1=ARG_TYPE     (I,L,R)         I   fortran type of the arg. |
|                                      I = Integer              |
|                                      L = Logical              |
|                                      R = Real                 |
|                                      Complex items must be    |
|                                      treated as 2 real items |
| $2=LOCAL_BYTES  (1,2,4,8,16)    I   length spec. of the arg. |
|                 or Y_DEF_BYTES        allowed ranges:         |
|                 or eval(2*Y_DEF_BYTES)  I/L: 1..Y_DEF_BYTES   |
|                                         R  : 4,8,16           |
| $3=TRANS_BYTES  (1,2,4,8,16)    I   length spec. for transfer|
|                                         I/L: 1,2,4,8          |
|                                         R  : 4,8,16           |
| $4=ARGUMENT     fortran_name    I   dummy argument           |
| $5=ARG_INDEX    empty or expr.  I   starting index, if the   |
|                                     dummy argument is declar-|
|                                     in the stub as an array. |
|                                     Example: A REAL*16 dummy |
|                                     argument must be declared|
|                                     as INTEGER(*) in the stub|
|                                     and the value 1 must be  |
|                                     used here.               |
|                                                               |
```

```
-----------------------------------------------------------------

-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MARSHALL_VAR_PART                              |
|                                                               |
|                                                               |
| Task           Finish of the fixed part and beginning of      |
|                the variable part.                             |
|                                                               |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          MARSHALL_ARRAYN                                |
|                                                               |
|                                                               |
| Task           This macro converts and sends a fortran array.|
|                It can be used only in the variable part.      |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=DIMENSIONS   (1,2,3,4,...)   I   number of dimensions      |
| $2=ARG_TYPE     (L,I,R)         I   fortran type of the arg.  |
| $3=LOCAL_BYTES  (1,2,4,8,16,..) I   length spec. of the arg.  |
| $4=TRANS_BYTES  (1,2,4,8,16,..) I   length spec. for transfer|
| $5=ARGUMENT     fortran_name    I   dummy argument            |
| $6=START_INDEX  integer_expres. I   starting index, of        |
|                                     declared array, e.g. 1    |
| $7=ARG_INDEX_1  integer_expres. I   starting index of transf.|
|                                     range, typically 0        |
| $8=ARG_LENGTH_1 integer_expres. I   number of items to be     |
|                                     transferred in first dim.|
| if DIMENSIONS=2 :                                             |
| $9=ARG_DECLAR_1 integer_expres. I   declared length of 1. dim|
| $10=ARG_INDEX_2 integer_expres. I   starting index of transf.|
|                                     range of 2. dimension     |
| $11=ARG_LEN_2   integer_expres. I   number of items to be     |
|                                     transferred in 2. dim.    |
| if DIMENSIONS=3 : ...                                         |
|                                                               |
| see MARSHALL_ITEM for further details                         |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
```

```
|                                                                |
| Macro          MARSHALL_STRING                                 |
|                                                                |
|                                                                |
| Task           This macro sends a fortran string.              |
|                It can be used only in the variable part.       |
|                                                                |
| Parameter       Type            I/O Usage                      |
| $1=ARGUMENT     fortran_name    I   dummy argument             |
| $2=ARG_INDEX    integer_expres. I   starting index in the      |
|                                     string                     |
| $3=ARG_LENGTH   integer_expres. I   number of characters       |
|                                                                |
------------------------------------------------------------------


------------------------------------------------------------------
|                                                                |
|                                                                |
| Macro          MARSHALL_ALIGNMENT                              |
|                                                                |
|                                                                |
| Task           In part 12 of the protocol there must be        |
|                sent 0 to 7 bytes to guarantee the alignment    |
|                on a 8 byte boundary.                           |
|                                                                |
| Parameter       Type            I/O Usage                      |
| (None)                                                         |
|                                                                |
------------------------------------------------------------------


------------------------------------------------------------------
|                                                                |
|                                                                |
| Macro          MARSHALL_SEND_CALL                              |
|                                                                |
|                                                                |
| Task           Flushing the Send Buffer in a Client Stub       |
|                                                                |
| Parameter       Type            I/O Usage                      |
| (None)                                                         |
|                                                                |
------------------------------------------------------------------


------------------------------------------------------------------
|                                                                |
|                                                                |
| Macro          MARSHALL_BUFFERED_CALL                          |
|                                                                |
|                                                                |
| Task           Not Flushing the Send Buffer in a Client Stub   |
|                                                                |
| Parameter       Type            I/O Usage                      |
```

```
| (None)                                                       |
|                                                              |
----------------------------------------------------------------


----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          MARSHALL_SEND_RETURN                          |
|                                                              |
|                                                              |
| Task           Flushing the Send Buffer in a Server Stub     |
|                                                              |
| Parameter       Type            I/O Usage                    |
| (None)                                                       |
|                                                              |
----------------------------------------------------------------


----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          MARSHALL_END                                  |
|                                                              |
|                                                              |
| Task           End of marshalling part                       |
|                                                              |
| Parameter       Type            I/O Usage                    |
| (None)                                                       |
|                                                              |
----------------------------------------------------------------
```

15.5.13 Unmarshalling Macros

```
----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          UNMARSHALL_FIX_PART                           |
|                                                              |
|                                                              |
| Task           Initialization of the fixed part.             |
|                                                              |
| Parameter       Type            I/O Usage                    |
| $1=NUM_OF_BYTES integer_value   I   must be reserved in the  |
|                                     send buffer              |
|                                                              |
----------------------------------------------------------------


----------------------------------------------------------------
|                                                              |
|                                                              |
| Macro          UNMARSHALL_HEADER                             |
|                                                              |
```

```
|                                                               |
| Task           Receive and Compute the header item            |
|                in a Client Stub. (Parts of the Header         |
|                are already computed by block and              |
|                module dispatcher.)                            |
|                                                               |
| Parameter       Type            I/O Usage                     |
| (none)                                                        |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          UNMARSHALL_INVOCATION                          |
|                                                               |
|                                                               |
| Task           Receive and Compute the header item            |
|                in a Server Stub.                              |
|                                                               |
| Parameter       Type            I/O Usage                     |
| $1=OPCODE       integer_value   I   rpc operation code        |
| $2=SYNC_MODE    (RPC_Synchron_Call, synchronisation mode      |
|                  RPC_Asynchron_Call)                          |
| $3=OP_NAME      text            I   rpc operation name        |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
| Macro          UNMARSHALL_ITEM                                |
|                                                               |
|                                                               |
| Task           Receive and Convert one dummy argument         |
|                                                               |
| Parameter       Type            I/O Usage                     |
|                                                               |
| $1=ARG_TYPE     (I,R,L)         I   fortran type of the arg.  |
| $2=LOCAL_BYTES  (1,2,4,8,16,..) I   length spec. of the arg.  |
| $3=TRANS_BYTES  (1,2,4,8,16,..) I   length spec. for transfer|
| $4=ARGUMENT     fortran_name    I   dummy argument            |
| $5=ARG_INDEX    empty or expr.  I   starting index, if the    |
|                                     dummy argument is an arr.|
|                                                               |
| see MARSHALL_ITEM for further details                         |
|                                                               |
-----------------------------------------------------------------


-----------------------------------------------------------------
|                                                               |
|                                                               |
```

```
| Macro          UNMARSHALL_VAR_PART                             |
|                                                                |
|                                                                |
| Task           Finish of the fixed part and beginning          |
|                of the variable part.                           |
|                                                                |
| Parameter       Type            I/O Usage                      |
| (none)                                                         |
|                                                                |
------------------------------------------------------------------


------------------------------------------------------------------
|                                                                |
|                     (Macro     header)                         |
|                     ------------------                         |
|                                                                |
| Macro          UNMARSHALL_ARRAYN                               |
|                                                                |
|                                                                |
| Task           This macro converts and stores a fortran array|
|                It can be used only in the variable part.       |
|                                                                |
| Parameter       Type            I/O Usage                      |
| $1=DIMENSIONS   (1,2,3,4,...)   I   number of dimensions       |
| $2=ARG_TYPE     (L,I,R)         I   fortran type of the arg.   |
| $3=LOCAL_BYTES  (1,2,4,8,16,..) I   length spec. of the arg.   |
| $4=TRANS_BYTES  (1,2,4,8,16,..) I   length spec. for transfer|
| $5=ARGUMENT     fortran_name    I   dummy argument             |
| $6=START_INDEX  integer_expres. I   starting index, of         |
|                                     declared array, e.g. 1     |
| $7=ARG_INDEX_1  integer_expres. I   starting index of transf.|
|                                     range, typically 0         |
| $8=ARG_LENGTH_1 integer_expres. I   number of items to be      |
|                                     transferred in first dim.|
| if DIMENSIONS=2 :                                              |
| $9=ARG_DECLAR_1 integer_expres. I   declared length of 1. dim|
| $10=ARG_INDEX_2 integer_expres. I   starting index of transf.|
|                                     range of 2. dimension      |
| $11=ARG_LEN_2   integer_expres. I   number of items to be      |
|                                     transferred in 2. dim.     |
| if DIMENSIONS=3 : ...                                          |
|                                                                |
| see MARSHALL_ITEM for further details                          |
|                                                                |
------------------------------------------------------------------


------------------------------------------------------------------
|                                                                |
|                                                                |
| Macro          UNMARSHALL_STRING                               |
|                                                                |
|                                                                |
| Task           This macro receives a fortran string.           |
```

```
|               It can be used only in the variable part.     |
|                                                             |
| Parameter       Type            I/O Usage                   |
| $1=ARGUMENT     fortran_name    I   dummy argument          |
| $2=ARG_INDEX    integer_expres. I   starting index in the   |
|                                     string                  |
| $3=ARG_LENGTH   integer_expres. I   number of characters    |
|                                                             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
|                                                             |
| Macro         UNMARSHALL_ALIGNMENT                          |
|                                                             |
|                                                             |
| Task          In part 12 of the protocol there must be      |
|               sent 0 to 7 bytes to guarantee the alignment  |
|               on a 8 byte boundary.                         |
|                                                             |
| Parameter       Type            I/O Usage                   |
| (None)                                                      |
|                                                             |
---------------------------------------------------------------
```

```
---------------------------------------------------------------
|                                                             |
|                                                             |
| Macro         UNMARSHALL_END                                |
|                                                             |
|                                                             |
| Task          End of marshalling part                       |
|                                                             |
| Parameter       Type            I/O Usage                   |
| (None)                                                      |
|                                                             |
---------------------------------------------------------------
```

Danksagung

Wir möchten allen, die zum Gelingen dieses Projekt beitrugen, für ihr Engagement danken. Herrn Prof. Dr.-Ing. R. Rühle danken wir für seine Unterstützung. Herrn Hartmut Aichele danken wir für die prototypische Programmierung der bsd-socket Kommunikation, Herrn Björn Lemke für die Entwicklungsarbeit am Monitortool, Frau Katrin Dieterle und Herrn Werner Dax (IKE) für die Portierung eines Teiles der Software auf VMS, Herrn Tham Do-Dinh für die Programmierung der Konvertierungsmakros, Herrn Matthias Glass für die Entwicklung des IPSM, Herrn Michael Grollmuss für Untersuchungen zum Locking, Herrn Markus Wenzel für die Erstellung des Testbeispiels, Herrn Thomas Wolf für die Entwicklung des Transport-Interfaces. Den Pilotanwendern Frau Heinke Höck (sie entwickelte das Beispiel ua1), Herrn Hans Pfeiffenberger, Herrn Chris Dodge und Herrn Roger Rohde vom Alfred-Wegener-Institut, Bremerhaven, Herrn Ralf Allrutz (Thema Parallelisierung), Herrn Axel Geyer und Herrn Dr. Thomas Ertl von der Theoretischen Astrophysik, Universität Tübingen, Herrn Dr.-Ing. Fritz Schmidt, Herrn Werner Dax und Herrn Johannes Kaltenbach vom IKE und Herrn Prof. Dr.-Ing B.-H. Kröplin und Herrn Michael Trüber und Herrn Stefan Weihe (Thema Common-Blöcke) vom ISD, Universität Stuttgart, danken wir für ihre Kritik und Anregungen und ihre Ausdauer beim ersten Einsatz und Test dieser Software. Außerdem sei Mark Reinhardt und Tim Kirby für ihre Mithilfe bei der Portierung des DFN-RPC auf CRAY-OSI gedankt.

Literatur

[1] Ralf Allrutz, Rolf Rabenseifner: Der DFN-RPC, ein Remote Procedure Call Tool. Proceedings, 15. DECUS München Symposium, 1992, S. 523-532.

[2] A.L. Ananda, B.H. Tay, E.K. Koh: ASTRA - An Asynchronous Remote Procedure Call Facility. In Proc. 11th Intl. Conf. on Distributed Computing Systems (ICDCS-11), IEEE, Arlington, Texas, USA, 1991, pp. 172-179.

[3] Developing DCE Applications for AIX and OS/2. IBM International Technical Support Center, Austin, July 1993, Chap. 12.0 Pipes.

[4] Al Geist et al.: PVM 3 User's Guide and Reference Manual. ORNL/TM-12187, Oak Ridge National Laboratory, Tennessee, 5/1993.

[5] David B. Johnson, Willy Zwaenepoel: The Peregrine High-performance RPC System. Software-Practice and Experience 23(2), 1993, pp. 201-221.

[6] Frank Kirschner, Lothar Ehnis: /sw – Das neue Software-Installationskonzept. Benutzerinformation des Rechenzentrums der Universität Stuttgart, Okt. 1993 (BI.10 93), S. 22-30.

[7] Informationen zu /sw im World Wide Web (WWW):
http://www.uni-stuttgart.de/Cis/ZentraleEinrichtungen/Rus/Bi/BI_html/BI93/10/File6.html
und http://www.uni-stuttgart.de/afs/rus/sw/doc/Soft.html/software_homepage.html

[8] Werner Kollak: Distributing FORTRAN Applications using DCE RPC. White Paper. Rechenzentrum Universität Stuttgart, March 5, 1993.

[9] Barbara Liskov, Liuba Shira: Promises: Linguistic Support for Efficient Asynchronous Procedure Calls in Distributed Systems. ACM SIGPLAN Conf. on Programming Language Design and Implementation 1988, pp. 260-267.

[10] Barbara Liskov, T. Bloom, D. Gifford, R. Scheifler, W.E. Weihl: Communication in the Mercury System. Proc. 21st Annual Hawaii Conference on System Science, Jan. 1988, pp. 178-187.

[11] Roland Laifer, Andreas Knocke: fidl – a tool for using DCE from Fortran. In Alexander Schill (Ed.), DCE - The OSF Distributed Computing Environment, International DCE Workshop, Proceedings, Karlsruhe, Germany, Oct. 7-8, 1993, pp. 78-88.

[12] Bruce-Jay Nelson: Remote Procedure Call. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsilvania, May 3, 1981.

[13] Open Software Foundation (OSF): OSF DCE Version 1.0, DCE Application Development Guide, Part 3: DCE Remote Procedure Call (RPC). Revision 1.0, Dec. 31, 1991.

[14] Rolf Rabenseifner, Werner Kollak: DFN-RPC im Vergleich mit DCE, ONC und PVM. DFN-Mitteilungen Nr. 34, März 1994, S. 18f.

[15] Rolf Rabenseifner, Armin Schuch: Comparison of DCE RPC, DFN-RPC, ONC and PVM. In Alexander Schill (Ed.), DCE - The OSF Distributed Computing Environment, International DCE Workshop, Proceedings, Karlsruhe, Germany, Oct. 7-8, 1993, pp. 39-46.

[16] Rolf Rabenseifner, Hans Dieter Reimann: Verteilte Anwendungen mit dem DFN-RPC. DFN Mitteilungen Nr. 31, März 1993, S. 23-25.

[17] Rolf Rabenseifner, Hans Dieter Reimann: Verteilte Anwendungen über Hochgeschwindigkeitsdatenkommunikation, der DFN Remote Procedure Call. Benutzerhandbuch, Rel. 1.0, Rechenzentrum Universität Stuttgart, 23.11.1992.

[18] R. Rabenseifner, H. D. Reimann: Erweitertes Remote Procedure Call Tool für verteilte Anwendungen. Aufgabenbeschreibung zum DFN-Projekt Rechenzentrum der Universität Stuttgart (RUS). 1992.

[19] Rabenseifner, R.: Distributed Applications between Workstation and Supercomputer using ISO/OSI Protocols. Proceedings, Twenty-Seventh Semi-Annual Cray User Group Meeting, London. April 22-26, 1991, pp. 80-84.

[20] Rolf Rabenseifner, Hans Dieter Reimann: Konzeption eines Remote Procedure Call Service für verteilte Anwendungen, Meilensteinbericht. Rechenzentrum der Universität Stuttgart (RUS). 1989.

[21] H. D. Reimann: Konzeption eines Monitortools für verteilte Anwendungen. DFN-Projektbericht. Rechenzentrum der Universität Stuttgart (RUS). 1993.

[22] Robert W. Scheifler, MIT Laboratory for Computer Science and Jim Gettys, Digital Equipment Corporation und MIT Project Athena: The X Window System. ACM Transactions on Graphics, Vol.5, No.2, April 1986, Pages 79-109.

[23] Alexander Schill: Remote Procedure Call: Fortgeschrittene Konzepte und Systeme – ein Überblick. Teil 2: Erweiterte RPC-Ansätze. Informatik Spektrum 15, pp. 145-155, Juni 1992.

[24] SUN microsystems: Network Programming Guide. Part Number 800-3850-00, 1990.

[25] Edward F. Walker, Richard Floyd, Paul Neves: Asynchronous Remote Operation Execution in Distributed Systems. In Proc. 10th Intl. Conf. on Distributed Computing Systems (ICDCS-10), IEEE, Paris, France, 1990, pp. 253-259.

[26] Yi-hsiu Wei and Chuan-lin Wu: Integrating RPC and Message Passing for Distributed Programming. In Alexander Schill (Ed.), DCE - The OSF Distributed Computing Environment, International DCE Workshop, Proceedings, Karlsruhe, Germany, Oct. 7-8, 1993, pp. 192-206.

Sämtliche Informationen und Publikationen zum DFN-RPC sind im WWW per Mosaic oder auch per anonymous ftp zu erreichen unter:

ftp://ftp.rus.uni-stuttgart.de/pub/rus/dfn_rpc/README_dfnrpc.html

Index