Janus/RSC

Kommunikation SAP R/3 mit Tandem Pathway Funktionsbeschreibung

Datum: 15.03.1999

Marcus von Cube
B & S GmbH
Borsigstraße 12
65205 Wiesbaden
Tel: 06122 / 7705-0
Fax: 06122 / 7705-25

Inhaltsverzeichnis

1 Einführung

1.1 Funktionsumfang

Die vorliegende Software Jnaus/RSC stellt eine Schnittstelle zwischen SAP R/3 und Tandem Pathway dar. Sie läuft innerhalb des SAP-Netzwerkes auf einem Unix-Host oder NT-Server. In der vorliegenden Fassung werden folgende Funktionen unterstützt:

- Steuerung über CPI-C und Remote Server Call (RSC);
- Auswahl der RSC-Initialisierungsdatei durch SAP;
- Verbindungsaufbau von SAP zum Tandemsystem;
- Transaktionssteuerung: BEGIN, END und ROLLBACK;
- Automatischer Transaktionsbegin und -ende für Einzel- und Mehrfachtransaktionen;
- WRITEREAD zu Tandemprozess oder Pathway-Server;
- Einheitliche Fehlerbehandlung.

Die Initiative geht in dieser Konstellation immer von SAP aus.

1.2 SAP-Softwareschnittstelle CPI-C

Die Kommunikation zwischen SAP und dem SAP_RSC-Gateway erfolgt über CPI-C. CPI-C-Aufrufe können von beliebigen Anwendungsprogrammen im SAP-Netzwerk durchgeführt werden, insbesondere auch von ABAP/4-Reports. SAP R/3 bildet das CPI-C-Protokoll innerhalb des Netzwerkes auf TCP/IP ab. Die darüber liegende Softwareebene merkt davon aber nichts.[1]

Über die Verbindungsaufbaufunktionen CMINIT und CMALLOC (bzw. COMMUNICATION INIT und COMMUNICATION ALLOCATE in ABAP/4) stellt das Anwendungsprogramm eine Verbindung zum SAP_RSC-Gateway her. Innerhalb einer solchen, in CPI-C *Conversation* genannten Verbindung laufen alle Transaktionen mit dem Tandem-System ab.

1.3 Tandem-Kommunikationsschnittstelle RSC

Die gesamte Kommunikation zum Tandem-System, insbesondere zu Pathway-Servern, erfolgt über das Tandemprodukt *Remote Server Call* (RSC.) RSC stellt eine vom darunterliegenden Kommunikationsprotokoll unabhängige Schnittstelle zu Pathway oder beliebigen Servern dar. RSC läßt sich über TCP/IP oder asynchrone Schnittstellen betreiben.

RSC unterstützt WRITEREAD und die Transaktionssteuerung über TMF. Pathway-Server sind kontextfrei, das heißt, es gibt keine feste Beziehung vom Anwendungsprogramm (Client) zu einem definierten Serverprozess. Kontextinformationen müssen also in jeder Message mit ausgetauscht werden.

Die CPI-C-Conversation stellt nur den Kontext für die Transaktionssicherung zur Verfügung!

1 CPI-C wurde von IBM ursprünglich für SNA-Netzwerke definiert, um die Programmierung von APPC-Verbindungen (LU 6.2) zu vereinfachen und zu vereinheitlichen. Bei Verbindungen von SAP R/2 auf IBM-Mainframes zu externen Systemen basiert CPI-C auf SNA.

2 Messageaufbau

Die Beschreibungen erfolgen im ABAP/4-Format und im C-Format.

2.1 Datenformate

2.1.1 Textfelder

Alle Textfelder sind mit Space aufgefüllt. Das gilt natürlich nicht für die Message-Nutzdaten. Deren Format legt alleine die Anwendung fest.

2.1.2 Zahlenfelder

Alle Zahlenfelder sind linksbündig mit Nullen gefüllt. Zahlen sind in der Regel fünfstellig. Links kann ein negatives Vorzeichen stehen.

2.2 Header

Der allgemeine Message-Header hat eine Länge von 12 Byte. Er entspricht dem von SAP geforderten Protokollkopf für die Anmeldeprozedur. Nur das Feld REQUEST_ID wird benutzt, die anderen enthalten Standard-Werte, die nicht ausgewertet werden.

2.2.1 Header im ABAP/4-Format

```
DATA: BEGIN OF MESSAGE_HEADER,
         REQ_ID(4),
         TYPE(4),
         MODE(1) TYPE N,
         RESERVED(3),
      END OF MESSAGE_HEADER.
```

2.2.2 Header im C-Format

```
typedef struct _message_header {
     char req_id[ 4 ];
     char type[ 4 ];
     char mode;
     char reserved[ 3 ];
} MESSAGE_HEADER;
```

2.3 Negative Quittungen

Negative Quittungen enthalten als REQ_ID den Text 'FREE' oder 'ERR!'. Das Feld TYPE bleibt leer. Der Text 'FREE' entspricht den SAP-Konventionen. Er ist mit einem Verbindungsabbruch gekoppelt und wird zum Beispiel bei einem fehlerhaften Logon erzeugt.

Der Text 'ERR!' beschreibt Fehlersituationen auf Anwendungsebene. Die Verbindung bleibt bestehen.

2.3.1 Quittungen im ABAP/4-Format

```
DATA: BEGIN OF ACK_MESSAGE,
         REQ_ID(4) VALUE 'FREE',                " oder 'ERR!'
         TYPE(4) VALUE 'CPIC',
         MODE(1) TYPE N VALUE 1,
         RESERVED(3) VALUE '   ',
         ERROR(5) TYPE N,
         ERROR_TEXT(512),
         UNUSED(3),                             " Auffüllung auf Vielfaches von 4 Bytes
       END OF ACK_MESSAGE.
```

Das Feld ERROR wird bei einem Logon-Fehler von SAP zum Beispiel mit "00027" gefüllt. Das Feld ERROR_TEXT enthält in diesem Fall Klartext.

2.3.2 Quittungen im C-Format

```
typedef struct _ACK_message {
     MESSAGE_HEADER header;
     char error[ 5 ];
     char error_text[ 512 ];
     char unused[ 3 ];
} ACK_MESSAGE;
```

2.3.3 Negative Quittungen bei RSC-Fehlern

Die folgende Quittung wird benutzt, wenn ein Fehler durch die RSC-API gemeldet wurde. Die Fehlernummer ist dann ERR_REMOTE (siehe unten). RSC_ERROR_INFO ersetzt das Feld ERROR_TEXT der negativen Quittung:

```
DATA: BEGIN OF RSC_ERROR_INFO,
         ERROR(5) TYPE N,
         SUBSYSTEM(5) TYPE N,
         SUBSYSTEM_ERROR(5) TYPE N,
         EXTENDED_ERROR(5) TYPE N,
         ERROR_CLASS(5) TYPE N,
         ERROR_TEXT(256) TYPE N,
       END OF RSC_ERROR_INFO.
```

Die Werte für ERROR_NO bis ERROR_CLASS sind Zahlen, die in der RSC-Dokumentation und der Datei RSC.H bzw. der ABAP/4-Includedatei ZBSRSCIE beschrieben sind. Aus ERROR-CLASS läßt sich beispielsweise entnehmen, ob die Aktion sinnvoll wiederholt werden kann oder nicht.

2.4 Positive Quittungen

Alle positiven Quittungen enthalten als REQ_ID den Text 'APPC' und in TYPE 'CPIC'. Dies entspricht den SAP-Konventionen.

Positive Quittungen können auch einen erläuternden Text enthalten. Dieser folgt auf eine "Fehlernummer" '00000', damit das Quittungsformat einheitlich bleibt.

2.5 Initialisierung der RSC-Verbindung

Die Verbindung zum Tandemsystem wird mit einer INIT-Message initialisiert. Die `REQ_ID` im Header ist '`INIT`'. `BUFFER` enthält den Namen der RSC.INI-Datei, die für die RSC-Initalisierung benutzt werden soll. In dieser Datei sind die RSC-Optionen, wie das Zielsystem oder der PATHMON-Name gespeichert.

SAP_RSC versucht mit den Parametern aus RSC.INI eine Verbindung zum Tandemsystem herzustellen. Die Antwort ist entweder eine positive oder eine negative Quittung.

2.5.1 Initialisierung im ABAP/4-Format

```
DATA: BEGIN OF INIT_MESSAGE,
         REQ_ID(4) VALUE 'INIT',
         TYPE(4) VALUE 'CPIC',
         MODE(1) TYPE N VALUE 1,
         RESERVED(3) VALUE '   ',
         BUFFER(512),
      END OF INIT_MESSAGE.
```

2.5.2 Initialisierung im C-Format

```
typedef struct _init_message {
     MESSAGE_HEADER header;
     char buffer[ 512 ];
} INIT_MESSAGE;
```

2.6 Datensätze von SAP zu Tandem

Datensätze tragen im Header die `REQ_ID` '`DATA`'. Das Feld `DESTINATION` enthält das Ziel der Message, im allgemeinen ein Pathway-Servername. Es kann auch ein beliebiger Prozessname angegeben werden.

Die Struktur wird nur in der tatsächlich erforderlichen Länge übertragen. Jede Message löst einen *RscWriteRead*-Aufruf aus. Erlaubt sind DATA-Messages erst nach erfolgreicher INIT-Message.

Die Antwort ist entweder eine REPLY-Message oder eine negative Quittung.

2.6.1 Datensätze im ABAP/4-Format

```
DATA: BEGIN OF DATA_MESSAGE,
         REQ_ID(4) VALUE 'DATA',
         TYPE(4) VALUE 'CPIC',
         MODE(1) TYPE N VALUE 1,
         RESERVED(3) VALUE '   ',
         DESTINATION(36),
         SIZE(5) TYPE N,
         BUFFER(31000),
       END OF DATA_MESSAGE.
```

2.6.2 Datensätze im C-Format

```
typedef struct _data_message {
     MESSAGE_HEADER header;
     char destination[ 36 ]
     char size[ 5 ];
     char buffer[ 31000 ];
} DATA_MESSAGE;
```

2.7 Antwortsätze von Tandem an SAP

Antwortsätze tragen im Header die `REQ_ID` 'REPL'. Sie haben ein ähnliches Format, wie die DATA-Messages. Sie werden als Antwort auf diese übermittelt. Anstelle einer REPLY-Message kann auch eine negative Quittung gesendet werden.

Das Feld `TRANSACTION` meldet den augenblicklichen Transaktionsstatus an den Aufrufer zurück. Es kann die Werte 0 (keine Transaktion) oder 1 (eine Transaktion ist offen) annehmen.

Die Struktur wird nur in der tatsächlich erforderlichen Länge übertragen.

2.7.1 Antwortsätze im ABAP/4-Format

```
DATA: BEGIN OF REPLY_MESSAGE,
         REQ_ID(4),
         TYPE(4),
         MODE(1) TYPE N,
         RESERVED(3),
         TRANSACTION(1),
         UNUSED(35),
         SIZE(5) TYPE N,
         BUFFER(31000),
      END OF REPLY_MESSAGE.
```

2.7.2 Antwortsätze im C-Format

```
typedef struct _reply_message {
     MESSAGE_HEADER header;
     char transaction;
     char unused[ 35 ];
     char unused[ 5 ];
     char buffer[ 31000 ];
} REPLY_MESSAGE;
```

2.8 Transaktionssteuerung

Die Transaktionssteuerung erfolgt über Messages mit der `REQ_ID` 'TRAN'. Das Feld `CODE` enthält einen Funktionscode, die Felder `SUCCESS_LO` und `SUCCESS_HI` enthalten Grenzwerte für den automatischen Transaktionsabschluß.

2.8.1 Transaktionssteuerung im ABAP/4-Format

```
DATA: BEGIN OF TRANSACT_MESSAGE,
         REQ_ID(4) VALUE 'TRAN',
         TYPE(4) VALUE 'CPIC',
         MODE(1) TYPE N VALUE 1,
         RESERVED(3),
         CODE(1),
         UNUSED(1) VALUE ' ',
         SUCCESS_LO(5) TYPE N,
         SUCCESS_HI(5) TYPE N,
       END OF TRANSACT_MESSAGE.
```

2.8.2 Transaktionssteuerung im C-Format

```
typedef struct _transaction_message {
     MESSAGE_HEADER header;
     char code;
     char unused;
     char success_lo[ 5 ];
     char success_hi[ 5 ];
} TRANSACTION_MESSAGE;
```

2.8.3 Transaktionscodes

Die folgende Tabelle enthält alle gültigen Transaktionscodes:

Code	*Name*	*Erklärung*
'N'	TT_NONE	Keine Transaktion gefordert oder aktiv.
'B'	TT_BEGIN	Mit der nächsten Daten-Message beginnt eine Transaktion.
'E'	TT_COMMIT	Die Transaktion wird sofort beendet.
'R'	TT_ROLLBACK	Die Transaktion wird sofort abgebrochen.
'S'	TT_SINGLE	Die nächste Data-Message wird in eine Transaktion gekapselt. `SUCCESS_LO` und `SUCCESS_HI` entscheiden, ob die Transaktion beendet wird. Das erste Wort des Server-Reply wird mit den beiden Werten binär verglichen. Fällt der Wert in das Intervall (einschließlich der beiden Grenzwerte), gilt die Transaktion als erfolgreich. Andernfalls bleibt die Transaktion offen und kann von der Anwendung beendet oder abgebrochen werden. Erkennbar ist dies am Feld `TRANSACTION` der Antwort-Message. Bei einem Erfolg wird es auf 0, sonst auf 1 gesetzt.
'A'	TT_AUTOCOMMIT	Die nächste Data-Message beendet eine Transaktion, die bereits besteht. Die Vorgehensweise ist wie bei SINGLE.

2.9 Fehlercodes

Fehlercodes haben einen symbolischen Namen, wie er im Programm verwendet wird, und einen numerischen Wert. Die folgende Tabelle enthält Fehlercodes, die in negativen Quittungen oder auch nur intern vorkommen können:

Symbolischer Name	*Wert*	*Erläuterung*
ERR_RECORD_TOO_BIG	1	Der Datensatz hat eine zu große Länge.
ERR_BAD_PARAMETER	2	Ein Übergabeparameter zu einem Unterprogramm ist ungültig.
ERR_CANNOT_SEND	10	CPI-C kann nicht senden, weil der Host nicht auf Empfang ist.
ERR_CANNOT_DISCONNECT	11	CPI-C kann die Verbindung nicht auflösen (siehe auch 10.)
ERR_DISCONNECTED	12	CPI-C hat momentan keine Verbindung.
ERR_PROTOCOL	13	Die Messageformate sind nicht korrekt.
ERR_MEMORY	14	Das Programm kann einen Speicherbereich nicht anfordern.
ERR_SAP_DOWN	100	Das SAP-System ist in Teilen nicht verfügbar.
ERR_REMOTE	200	RSC meldet einen Fehler.
ERR_NO_REMOTE_CONNECTION	201	INIT wurde noch nicht ausgeführt.
ERR_CPIC	300	CPI-C meldet einen Fehler (nur in ABAP/4)
ERR_FATAL	10000	Fataler Systemfehler.
ERR_65535	65535	Fehlernummer nicht korrekt.

3 ABAP/4-Zugriff auf Tandem-Server

3.1 Include-Reports

Um die APAB/4-Programmierung zu erleichtern, gibt es drei Include-Reports, die die notwendigen Datenstrukturen und Unterprogramme (FORMs) enthalten. Die Unterpgramme übernehmen den kompletten Message-Aufbau und Versand.

3.1.1 Datenstrukturen ZBSRSCID

Der Report ZBSRSCID muß am Anfang des Prgramms mit INCLUDE eingefügt werden. Er enthält alle Messages und zusätzliche Strukturen (Feldleisten) für die Unterprogramme. Der Report enthält auch die notwendigen INCLUDEs für die CPI-C-Kommunikation (RSEBCASC und RSCPICDF) und die RSC-Definitionen.

3.1.2 RSC-Definitionen ZBSRSCIE

Der Report ZBSRSCIE wird von ZBSRSCID automatisch geladen. Er enthält einen Auszug der Definitionen in RSC.H im ABAP/4-Format. Insbesondere sind alle Fehlercodes der Basis-RSC-API sowie die Fehlerklassen und Subsystemkennungen definiert.

3.1.3 FORM-Unterprogramme ZBSRSCIF

Der Report ZBSRSCIF muß *am Ende* des Programms mit INCLUDE eingfügt werden. Er enthält FORM-Unterprogramme, die in den folgenden Abschnitten beschrieben werden.

3.2 Beschreibung der Datenstrukturen

Der Report ZBSRSCID definiert alle Message-Datenstrukturen und deren Längen aus dem vorherigen Kapitel und zusätzlich Konstanten, Strukturen für den Kommunikationsstatus und RSC-Informationen:

Definition	*Erläuterung*
MESSAGE_HEADER HEADER_SIZE	Der von SAP definierte Header (12 Bytes.)
INIT_MESSAGE INIT_MESSAGE_SIZE	Message zum Aufbau der RSC-Verbindung.
DATA_MESSAGE DATA_MESSAGE_SIZE	Message zum Senden von Daten an Server. Diese Struktur hat die volle Länge (310053 Bytes.)
REPLY_MESSAGE REPLY_MESSAGE_SIZE	Antwort auf die DATA-Message. Diese Struktur hat die nur die Länge des Headers ohne Daten (53 Bytes.)
ACK_MESSAGE ACK_MESSAGE_SIZE	Positive oder negative Quittung.
TRANSACT_MESSAGE TRANSACT_MESSAGE_SIZE	Steuerung von TMF-Transaktionen.
TT_NONE TT_BEGIN TT_COMMIT TT_SINGLE TT_AUTOCOMMIT	Transaktionscodes.
ERR_RECORD_TOO_BIG ERR_BAD_PARAMETER ERR_CANNOT_SEND ERR_CANNOT_DISCONNECT ERR_DISCONNECTED ERR_PROTOCOL ERR_REMOTE ERR_NO_REMOTE_CONNECTION ERR_CPIC ERR_FATAL	Fehlernummern.
RSC_ERROR_INFO	Aufschlüsselung der RSC-Fehlermeldungen.
COMM	Alle Statusinformationen der laufenden Kommunikation.

3.2.1 RSC-Fehler in RSC_ERROR_INFO

Wird im Feld COMM-ERROR der Fehler ERR_REMOTE gemeldet, dann sind zusätzliche Informationen in der Struktur RSC_ERROR_INFO zu finden, die im Abschnitt über Quittungen beschrieben ist. Symbolische Namen für die möglichen Werte sind im Include-Report ZBSRSCIE verfügbar.

Feld	*Erläuterung*
ERROR	Die RSC-Fehlernummer.
SUBSYSTEM	Die Kennung des Subsystems, in dem der Fehler auftrat.
SUBSYSTEM_ERROR	Die Fehlernummer innerhalb des Subsystems.
EXTENDED_ERROR	Weitere Subsystem-Fehlerinformation.
ERROR_CLASS	Art des Fehlers, unter anderem die Information ob ein Retry möglich ist.
ERROR_TEXT	Fehlermeldung im Klartext. Diese ist auch in COMM-ERROR_TEXT zu finden.

3.2.2 Der Kommunikationsstatus COMM

Alle kommunikationsrelevanten Statusinformationen werden in der Feldleiste COMM gespeichert:

```
DATA: BEGIN OF COMM,
         SYMDEST(8) VALUE 'TANDEM',
         CONNECTED TYPE I VALUE 0,
         CONV_ID(8),
         STATUS(4) TYPE X,
         DATAINFO(4) TYPE X,
         ERROR(5) TYPE N,
         CPIC_ERROR LIKE SY-SUBRC,
         ERROR_TEXT(512),
         TRANSACTION(1) TYPE N,
         MESSAGE_TYPE(4),
         DATA_SIZE(5) TYPE N,
         REPLY_SIZE TYPE I,
       END OF COMM.
```

Feld	*Erläuterung*
SYMDEST	CPI-C-Name (aus TXCOM) des SAP_RSC-Gateway-Prozesses.
CONNECTED	1, wenn eine Verbindung besteht, 0 sonst.
CONV_ID	CPI-C Conversation-ID für die laufende Verbindung.
STATUS	Das STATUSINFO-Feld des letzten CPI-C-Empfangs-Aufrufs: CM_SEND_RECEIVED oder CM_NO_STATUS_RECEIVED.
DATAINFO	Das DATAINFO-Feld des letzten CPI-C-Empfangs-Aufrufs (wird nicht ausgewertet.)
ERROR	Der letze zurückgemeldete Fehlercode.
CPIC_ERROR	CPI-C-Returncode des letzten CPI-C-Aufrufs.
ERROR_TEXT	Klartext-Fehlermeldung.
TRANSACTION	1, wenn eine Transaktion *nicht* automatisch beendet werden konnte, 0 sonst.
MESSAGE_TYPE	Die letzte REQ_ID.
DATA_SIZE	Länge der Nutztdaten.
REPLY_SIZE	Länge der letzten Tandem-Antwort.

3.3 Beschreibung der Funktionen

Die Funktionen in ZBSRSCIF kapseln die Kommunikation mit dem SAP_RSC-Gateway vollständig. Die folgende Beschreibung enthält nur solche Funktionen, die im Hauptreport sinnvoll aufgerufen werden können. Andere hier nicht beschriebene Funktionen dienen nur als interne Hilfen.

Jede Funktion liefert einen Wert `RESULT` zurück, der im Report als `TYPE I` deklariert sein muß. Der Name dieser Variable ist natürlich frei wählbar, da sie als Parameter verwendet wird. `RESULT` nimmt nur die Werte 0 (OK) oder 1 (Fehler aufgetreten) an.

Messages, die an Tandem-Server geschickt werden sollen, müssen ebenfalls im Hauptreport deklariert werden. Ihr Aufbau muß der jeweiligen Tandem-Messagestruktur entsprechen.

Ergebnisse, Fehlermeldungen und gemeinsame Parameter werden in der Struktur `COMM` abgespeichert bzw. dieser entnommen. Bei RSC-Fehlern wird außerdem `RSC_ERROR_INFO` gefüllt.

3.3.1 CPIC_CONNECT

Aufruf:

```
PERFORM CPIC_CONNECT USING RESULT.
```

Die Funktion baut die CPI-C-Verbindung zum SAP_RSC-Gateway auf. Der symbolische Name des Gateways, so wie in TXCOM vereinbart, muß in das Feld `COMM-SYMDEST` eingetragen werden. Bei Erfolg wird `COMM-CONNECTED` auf 1 gesetzt und `COMM-CONV_ID` gefüllt.

Die Funktion muß zuerst aufgerufen werden.

3.3.2 CPIC_DISCONNECT

Aufruf:

```
PERFORM CPIC_DISCONNECT USING RESULT.
```

Die Funktion baut die CPI-C-Verbindung zum SAP_RSC-Gateway wieder ab. Bei Erfolg wird `COMM-CONNECTED` auf 0 gesetzt.

Die Funktion muß zuletzt aufgerufen werden, um den Gateway-Prozess zu beenden und die Resourcen freizugeben.

3.3.3 RSC_INIT

Aufruf:

```
PERFORM RSC_INIT USING 'RSC.INI' RESULT.
```

Die Funktion baut die Verbindung zwischen dem SAP_RSC-Gateway und dem Tandem-TDP auf. Alle Verbindungsparameter werden der Datei RSC.INI entnommen. Hier kann ein beliebiger Dateiname angegeben werden, auch mit Pfadangabe z. B. */user/pathway/myrsc.ini.*

Mindestens ein Aufruf der Funktion ist zwingend erforderlich.

Der Aufruf an `RSC_INIT` kann mehrfach innerhalb einer CPI-C-Verbindung erfolgen. Das Gateway baut die jeweils letzte RSC-Verbindung vorher ab. Offene TMF-Transaktionen werden dann allerdings verworfen.

3.3.4 RSC_WRITEREAD

Aufruf:

```
PERFORM RSC_WRITREAD USING SERVER
                           MESSAGE MESSAGE_LENGTH
                           REPLY   REPLY_LENGTH
                           RESULT.
```

Die Funktion sendet die Tandem-Servermessage MESSAGE an den Server SERVER und gibt die Antwort in REPLY zurück. Die Länge der Message ist MESSAGE_LENGTH und muß exakt stimmen. REPLY_LENGTH ist die maximal erwartete Antwortlänge vom Server. Die exakte Länge der Antwort wird in COMM-REPLY_SIZE abgelegt. Das Feld COMM-TRANSACTION wird auf 1 gesetzt, wenn eine TMF-Transaktion noch offen ist.

Bei Fehlern bleibt REPLY unverändert und RESULT ist 1. Ist die Antwort vom Server länger als REPLY_LENGTH, dann wird der Fehler ERR_RECORD_TOO_BIG gemeldet.

3.3.5 RSC_<BEGIN/COMMIT/ROLLBACK>_WORK

Aufruf:

```
PERFORM RSC_BEGIN_WORK    USING RESULT
PERFORM RSC_COMMIT_WORK   USING RESULT
PERFORM RSC_ROLLBACK_WORK USING RESULT
```

Die Funktionen dienen der expliziten TMF-Transaktionssteuerung. RSC_BEGIN_WORK führt allerdings erst beim nächsten RSC_WRITEREAD zu einer TMF-Transaktion. RSC_COMMIT_WORK und RSC_ROLLBACK_WORK setzen das Feld COMM-TRANSACTION auf 0.

3.3.6 RSC_<SINGLE/AUTOCOMMIT>_WORK

Aufruf:

```
PERFORM RSC_SINGLE_WORK     USING SUCCESS_LO SUCCESS_HI RESULT
PERFORM RSC_AUTOCOMMIT_WORK USING SUCCESS_LO SUCCESS_HI RESULT
```

Die Funktionen dienen der impliziten TMF-Transaktionssteuerung. Beide wirken erst beim nächsten RSC_WRITE-READ. RSC_SINGLE_WORK führt auf der Tandem vor dem WRITEREAD ein automatisches BEGINTRANSAC-TION aus, während RSC_AUTOCOMMIT_WORK eine offene Transaktion erwartet.

Der TDP auf der Tandem vergleicht das erste 16-Bit-Wort des Server-Reply gegen die beiden Werte SUCCESS_LO und SUCCESS_HI. Liegt die Antwort in diesen Grenzen (inklusive der Grenzwerte sebst), wird ENDTRANSACTION ausgelöst, die Transaktion also korrekt beendet. Im anderen Fall bleibt die Transaktion offen und muß vom Report mit RSC_COMMIT_WORK oder RSC_ROLLBACK_WORK beendert werden. Das Feld COMM-TRANSACTION ist in diesem Fall 1, bei erfolgreich beendeter Transaktion 0.

3.4 Programmbeispiel

```
REPORT ZBSRSCT LINE-SIZE 255.
************************************************************************
*                                                                      *
*  Verbindungsaufbau zur Tandem mit Hilfe von CPI-C und SAP_RSC        *
*                                                                      *
*  Testprogramm                                                        *
*                                                                      *
************************************************************************

*
*  RSC-Daten
*
INCLUDE ZBSRSCID.

*
*  Tandem-Messages
*
DATA: RESULT TYPE I,
      BEGIN OF MESSAGE,
         REPL_CODE(2) TYPE X VALUE '0000',
         TRANS_CODE(8) VALUE 'GZEIN010',
         MESS_INFO(200) VALUE '00A4711',
      END OF MESSAGE.

DATA: MESSAGE_SIZE TYPE I.
DESCRIBE FIELD MESSAGE LENGTH MESSAGE_SIZE.

DATA: REPLY(900).

DATA: REPLY_SIZE TYPE I.
DESCRIBE FIELD REPLY LENGTH REPLY_SIZE.

*
*  Verbindungsaufbau zum SAP_RSC-Gateway
*
PERFORM CPIC_CONNECT USING RESULT.
IF RESULT = 0.
    PERFORM RSC_INIT USING 'RSC.INI' RESULT.
ENDIF.
IF RESULT = 0.
    WRITE: / ACK_MESSAGE-ERROR_TEXT.
ENDIF.

*
*  Anfordern einer Einzelnen TMF-Transaktion
*
IF RESULT = 0.
    PERFORM RSC_SINGLE_WORK USING 0 0 RESULT.
ENDIF.
```

```
*
*  Testmessage zum Server senden
*
IF RESULT = 0.
    PERFORM RSC_WRITEREAD USING 'TEST-SERVER'
                                MESSAGE MESSAGE_SIZE
                                REPLY 900
                                RESULT.
ENDIF.
IF RESULT = 0.
    WRITE: / REPLY,
           / 'Reply-Size =', COMM-REPLY_SIZE,
           / 'Transaction =', COMM-TRANSACTION.
    PERFORM CPIC_DISCONNECT USING RESULT.
ENDIF.

*
*  Ausgabe des letzten (Fehler-)Status
*
WRITE: / 'result =', RESULT.
WRITE: / 'comm-error:', COMM-ERROR.
WRITE: / 'comm-error_text:', COMM-ERROR_TEXT.

*
*  RSC-Programme (FORM, muss nach hinten!)
*
INCLUDE ZBSRSCIF.
```

4 Umgebung

4.1 Installation unter Unix

Die Installation von SAP_RSC erfolgt auf einem SAP-Rechner, bevorzugt auf dem Gateway-System. Kopieren Sie die ausführbare Datei *sap_rsc* und das Script *sap_rsc.sh* in das Homeverzeichnis eines Users *janus*. Alle Dateien sollten diesem User gehören. Durch Setzen des *s*-Bits mit *chmod* läuft das Gateway nicht unter dem Account des SAP-Administrators sondern dem des Users *janus*. Erlauben sie dem SAP-Administrator zusätzlich die Ausführung des Scripts und der ausführbaren Datei.

Erfolgt die Installation nicht auf dem Gateway-System, so müssen Sie sicherstellen, daß das Gateway-System RSH-Aufrufe auf dem Rechner, auf dem Janus installiert ist, durchführen darf. Dies erfolgt zum Beispiel durch einen Eintrag in */etc/hosts.equiv*. Verschiedene Unix-Implementierungen unterscheiden sich hier. Auch der Name des *rsh*-Programms variiert und kann auch *rcmd* sein.

4.1.1 Shell-Script *sap_rsc.sh*

Der Start von *sap_rsc* erfolgt über das folgende Script, in dem Sie die Umgebungsvariablen setzen können:

```
#!/bin/sh
# Start des CPIC-RSC Gateways mit Umgebungsvariablen
#
cd /home/janus
TRACE=1; export TRACE
DEBUG=0; export DEBUG
CPIC_TRACE=0; export CPIC_TRACE

/home/janus/sap_rsc $*
```

4.1.2 RSC-Konfiguration

Im Janus-Verzeichnis oder an beliebiger anderer Stelle ist eine RSC.INI-Datei anzulegen, in der die Kommunikationsparameter für RSC abgelegt sind. Im RSC-Verzeichnis findet sich eine Beispieldatei. Der Name dieser Datei wird vom ABAP/4-Report an Janus übergeben.

Die ausführbaren RSC-Programme sollten einen Link in ein Verzeichnis, das im PATH liegt besitzen, zum Beispiel */usr/bin*.

4.1.3 Konfiguration im SAP-System

In die TXCOM-Tabelle ist eine Zeile einzutragen, die auf das externe Programm *sap_rsc.sh* zeigt. Der Protokolltyp ist 'E' (externes Programm.)

4.2 Installation unter Windows NT

Die Installation von SAP_RSC erfolgt auf einem SAP-Rechner. Da Windows NT keine Unterstützung für das RSH-Protokoll bietet, mit dem externe Programme auf anderen Systemen gestartet werden können, *muß* diese Installation auf einem System erfolgen, auf dem der SAP-Gateway-Prozess läuft. Der Start von Janus erfolgt dann durch direkten Programmaufruf.

Legen Sie ein Verzeichnis JANUS an, auf das der SAP-Administrator Zugriff hat. Alle Programme kopieren Sie in dieses Verzeichnis.

4.2.1 Batchdatei *start_sap_rsc.cmd*

Der Start von *sap_rsc* erfolgt über die folgende Batchdatei, in der Sie die Umgebungsvariablen setzen können:

```
@echo off
echo ***** SAP_RSC %1 %2 %3 ******
D:
cd \JANUS
set TRACE=1
set DEBUG=1
set CPIC_TRACE=1
del CPICTRC* 2>&1 >nul
sap_rsc.exe %1 %2 %3 2>&1 >con
```

4.2.2 Hilfsprogramm EXE2CMD

Da das Gateway nur *exe*-Programme starten kann, gehört ein kleines Hilfsprgramm *EXE2CMD* zum Lieferumfang, das es erlaubt beliebige Batchdateien mit Extension *cmd* zu starten. Dazu wird es genauso benannt wie die zu startende Batchdatei, aber mit Extension *exe*, in diesem Fall also *start_sap_rsc.exe*.

4.2.3 RSC-Konfiguration

Im Janus-Verzeichnis oder an beliebiger anderer Stelle ist eine RSC.INI-Datei anzulegen, in der die Kommunikationsparameter für RSC abgelegt sind. Im RSC-Verzeichnis findet sich eine Beispieldatei. Der Name dieser Datei wird vom ABAP/4-Report an Janus übergeben.

Die ausführbaren RSC-Programme sollten im Pfad stehen.

4.2.4 Konfiguration im SAP-System

In die TXCOM-Tabelle ist eine Zeile einzutragen, die auf das externe Programm *start_sap_rsc.exe* zeigt. Der Protokolltyp ist 'E' (externes Programm.)

4.3 Informationsquellen

4.3.1 CPI-C-Trace

Solange die Umgebungsvariable TRACE auf 1 oder "ON" gesetzt ist, erzeugen die CPI-C-Programme Dateien mit der Endung *.trc* in ihrem Arbeitsverzeichnis. In diesen Dateien findet sich ein Mitschnitt der gesamten CPI-C-Konversation des entsprechenden Programms. Die Dateien werden allerdings jedesmal überschrieben.

Die Umgebungsvariable CPIC_TRACE wird von der SAP-CPI-C-Bibliothek ausgewertet und erlaubt die detaillierte Verfolgung von Kommunikations-Problemen. Die erzeugten Dateien heißen CPICTRC... im Janus-Verzeichnis. Erlaubte Werte sind 0 bis 3.

4.3.2 Logging

Die Programme erzeugen für wichtige Ereignisse und Fehler Einträge in Log-Dateien mit der Endung *.log*. Die Dateien werden jedesmal fortgeschrieben und müssen von Zeit zu Zeit manuell bereinigt werden.

4.3.3 Konsolmeldungen

Von SAP_RSC wird bei schweren Fehlern der Befehl *meldung <level> <Textstring>* aufgerufen. Er muß im Pfad des Programms oder im Gateway-Homeverzeichnis vorliegen und kann ein beliebiges Script oder Programm sein. *<level>* nimmt Werte von 0 (Info) bis 3 (fatal) an. Schwere Fehler können so für eienen Alarm auf einer Operator-Konsole ausgewertet werden.