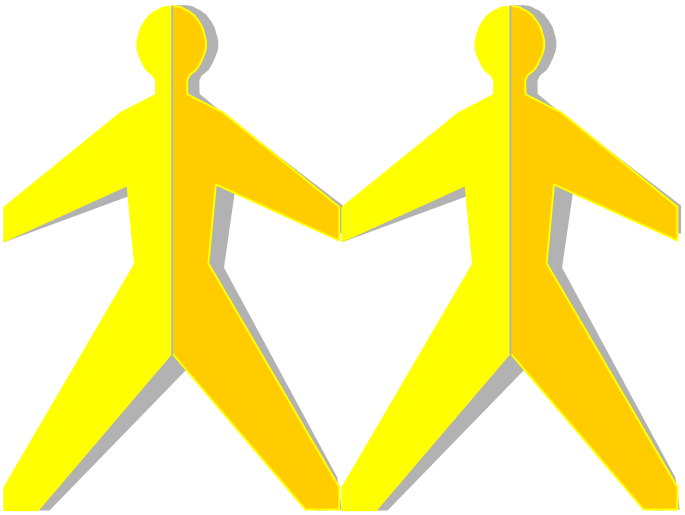

rvs®

rvscal

C-CAL-Schnittstelle

Beschreibung

rvs®

Version 1.0

Beschreibung

Pascalstraße 11

10587 Berlin

Inhaltsverzeichnis

Das vorliegende Dokument enthält eine Beschreibung der rvs®-Schnittstelle `rvscal`.

Zu Beginn des Dokuments finden Sie eine Übersicht der Repräsentationsmittel und Darstellungskonventionen, die im Text verwendet wurden.

Kapitel 2 bietet einen Gesamtüberblick zu allen rvs®-Schnittstellen.

Nach einer Beschreibung der Einbetteung der `rvscal`-Schnittstelle in die rvs®-Struktur im Kapitel 3, stellen Ihnen die Kapitel 4 bis 6 die Möglichkeiten, die Ihnen die `rvscal`-Schnittstelle bietet, sowie sämtliche Kommandos und Funktionen, im Detail vor.

1 Repräsentationsmittel und Darstellungskonventionen

Dieser Abschnitt enthält die Beschreibung, welche Darstellungskonventionen in diesem Handbuch verwendet werden und welche Bedeutung besonders gekennzeichnete Ausdrücke haben.

Darstellungskonventionen

`courier`	Kommandos, Menübefehle, Dateinamen, Pfadnamen, Programme, Beispiele, Script-Dateien, Optionen, Qualifiers, Datensätze, Felder, Modi, Fensternamen, Dialogboxen und Status
FETT und GROSSBUCHSTABIG	Parameter, Umgebungsvariablen, Variablen
"Hochkommata"	Verweise auf andere Handbücher, Kapitel und Abschnitte, Literatur
fett	wichtige Begriffe, Betriebssystemnamen, Eigennamen, Schaltflächen (Buttons), Funktionstasten

Begriffe

rvsX ist das Synonym für rvs® auf **UNIX** Systemen.

rvsNT ist das Synonym für rvs® auf **Windows NT** Systemen.

rvsXP ist das Synonym für rvs® auf **Windows XP** Systemen.

rvs400 ist das Synonym für rvs® auf **OS/400** Systemen.

Verzeichnisse

Weil Benutzerverzeichnisse auf unterschiedlichen Plätzen bei den unterschiedlichen Betriebssystemen zu finden sind, benutzen wir in diesem Handbuch die Variable **$RVSPATH**. Die Standardwerte sind:

- `/home/rvs/` für **AIX**, **Solaris**, **IRIX**, **Linux** und **SCO**
- `/users/rvs/` für **HP-UX**
- `/defpath/rvs/` für **SINIX**
- `C:\Programme\rvs`® für **Windows NT** und **Windows XP**

Ersetzen Sie diese Variable durch Ihren richtigen Pfad.

Dateinamen bei **OS/400** Systemen werden immer groß geschrieben.

2 rvs® und seine Schnittstellen im Überblick

rvs® bietet einen zuverlässigen Datenübertragungsdienst, in dem es als eigenständiges Werkzeug Daten senden, empfangen und verteilen kann. rvs® kann in Anwendungen für die Automatisierung des Datenaustausches zwischen Netzknoten und Benutzern integriert werden. Typische Anwendungsbereiche von rvs® sind EDI (**E**lectronic **D**ocument **I**nterchange), CAD (**C**omputer **A**ided **D**esign), Finanztransaktionssysteme und sichere Übertragung von Stammdaten und Daten von Medienkonzernen.

Für verschiedene Grade der Automatisierung sind passende Schnittstellen zu rvs® für die Erzeugung von Sende- und Empfangseinträgen verfügbar:

Dialog Schnittstelle `rvsdia`	Ist ein interaktives Werkzeug für die Erzeugung jeweils einzelner Einträge; Abfragefunktionen informieren Sie über den Status Ihrer Anforderungen.
Batch-Schnittstelle (auch: Kommandozeilen-Schnittstelle) `rvsbat`	Liest Kommandos aus einer Datei. Diese Eingabedatei ist eine einfache Textdatei, die mit einem beliebigen Editor vorbereitet oder als Ausgabedatei eines Anwendungsprogrammes erzeugt werden kann.
C-CAL-Schnittstelle `rvscal`	Erlaubt Anwendungsprogrammen durch Aufrufe von Funktionen der Programmiersprache C die direkte Generierung von rvs® Kommandoeinträgen.
J-CAL-Schnittstelle	Diese Java-Schnittstelle wurde im Zusammenhang mit dem rvs Client Server, einer netzwerkfähigen Erweiterung von rvs® portable, entwickelt.
XML-Schnittstelle	XML-Schnittstelle ermöglicht Export und Import von rvs® bezogenen Daten im XML-Format.

rvs® analysiert die Sendeeinträge, erzeugt Sendekommandos und startet die Sendeprozesse, die Ihre Dateien an den oder die richtigen Nachbarknoten im OFTP-Netzwerk für die Zustellung oder die Weiterleitung (Routing) übertragen. Der jeweilige Sendeeintrag bleibt im Wartezustand, bis eine positive Bestätigung vom Empfänger (EERP) kommt.

Empfangsprozesse nehmen eintreffende Dateien an und legen sie in temporären Dateien ab. Danach werden die Dateien dem lokalen Empfänger zugestellt oder wenn nötig, werden passende Sendeeinträge für die Weiterleitung an ferne Stationen erzeugt.

Die Zustellung eintreffender Dateien kann durch residente Empfangseinträge beeinflusst werden, die rvs® mitteilen, unter welchem Namen die Datei gespeichert werden soll und ob ein Job für die Weiterverarbeitung der eintreffenden Daten gestartet werden soll.

Jede der oben vorgestellten Schnittstellen erzeugt je einen Eintrag für jede Datei.

3 Die C-Cal-Schnittstelle `rvscal`

Die C-Cal-Schnittstelle `rvscal` kann direkt mit einem Benutzeranwendungsprogramm verbunden werden. Sie ermöglicht die Eingabe von Aufträgen in die rvs® Datenbank direkt aus dem Benutzeranwendungsprogramm. Ein Beispielprogramm in C und einige Kommandozeilen-Beispiele sind in der vorliegenden rvs® Version integriert.

4 Starten der C-Cal-Schnittstelle rvscal. Verwendete Parameter und korrespondierende rvs®-Kommandos

Dieses Kapitel beschreibt, wie Sie die C-Cal-Schnittstelle starten und welche Parameter Sie verwenden können. Außerdem erhalten Sie eine Beschreibung der korrespondierenden globalen Kommandos von rvs®. Die Syntax der Kommandos wird ebenso erklärt wie der Prototyp der Funktion `rvscal()`.

Für alle Aufgaben (Senden und Empfangen von Dateien ebenso wie die Administration von rvs®), die Sie mit rvs® zu erledigen haben, können Sie entweder

- Kommandos in eine Textdatei schreiben und an die Kommandozeilen-Schnittstelle (`rvsbat`) übergeben,
- Kommandos an die Funktion `rvscal()` der C-Cal-Schnittstelle übergeben oder
- C-Funktionen für die C-Cal-Schnittstelle benutzen (für jedes Kommando eine eigene Funktion)

Die Kommandos von `rvscal()` (C-Cal-Schnittstelle) sind identisch mit denen von `rvsbat` (Kommandozeilen-Schnittstelle) und werden im **Kapitel 5: Beschreibung der Kommandos** erläutert.

Die C-Funktionen der C-Cal-Schnittstelle sind im **Kapitel 6: Arbeiten mit C-Funktionen** beschrieben.

4.1 Die C-CAL-Schnittstelle verwenden

4.1.1 Kompilieren und binden der C-CAL-Schnitttelle für rvsNT

C-CAL-Schnittstelle ist getestet und geschrieben in Microsoft Visual C++. Die Prototypdefinitionen befinden sich in der Header-Datei `rvscal.h`. Die folgende Zeile muss das Anwendungsprogramm enthalten:

- `#include rvscal.h`

Es wird noch eine Header-Datei `rixstd.h` benötigt, die aber nicht explizit inkludiert werden muss. Wir empfehlen Run-Time-Linking zum Binden der C-CAL-Schnittstelle zu verwenden. Daher muss Ihre Anwendung folgende Aufrufe absetzen:

- `LOAD_RVSCALL_DLL (HANDLE &hlib)`. Das ist ein Makro zum Laden der korrekten Version der dynamischen Bibliothek von rvsNT. Dieser Makro ruft indirekt die Funktion `Load-Library()` und lädt so die aktuelle Version der rvsNT-Bibliothek. Der Parameter `&hlib` ist die Adresse des handles der DLL. Wir empfehlen diesen Makro zu benutzen, um kompatibel zu späteren Versionen der rvs-Biblithek sein zu können.
- `GetProcAddress()` .Mittels `GetProcAddress()` holt man sich die Adresse einer rvsNT-Funktion, die man aufrufen möchte z.B. `rvsCreateSendEntry()`.
- `FreeLibrary()` sollten Sie aufrufen, nachdem Die die Benutzung von der C-CAL-Schnittstelle beendet haben.

4.1.1.1 Optionen zum Kompilieren und Binden

- **Packing**: Die structs der C-CAL-Schnittstelle werden an 4-Byte-Grenzen ausgerichtet. Daher muss die Anwendung mit der Option `/Zp4` kompiliert werden.
- **char type**: Die C-CAL-Schnittstelle benutzt `unsigned char` als default char type. Daher muss auch die Applikation mit der dies bewirkenden Kompilieroption `/J` kompiliert werden.

4.1.1.2 Bibliothekdateien

Für Ihre Anwendung, die C-CAL-Schnittstelle benutzt, benötigen Sie zwei dynamischen Bibliotheken von rvsNT:

- `RVSCALL.DLL` , die von der Anwendung geladen wird.
- `RVSCALL22.DLL`, die zur Laufzeit von `RVSCALL.DLL` geladen wird; darf nicht von der Applikation geladen werden.

Beispiel:

```
#include <windows.h>
#include <stdio.h>
#include "rvscal.h"
void main(void)
{
    HANDLE  hRvsLib;
    char        str[128];
    FARPROC     prvsGetDBVersion;

    LOAD_RVSCAL_DLL(&hRvsLib);
    if (!hRvsLib)
    {
            printf("Error in LoadLibrary, rc= %d\n", Get-
LastError());
            return;
        }

    prvsGetDBVersion = GetProcAddress( hRvsLib,
"rvsGetDBVersion");
    if (!prvsGetDBVersion)
    {
            printf("Error in GetProcAddress rvsGetDBVer-
sion, rc= %d\n",
GetLastError() );
            return;
        }

    prvsGetDBVersion(str);
    printf("Version: %s\n", str);
}
```

4.1.2 C-CAL-Schnittstelle für UNIX und OS/400

Für den Einsatz der C-CAL-Schnittstelle müssen Sie folgende Bibliothek binden:

- `rpulib.a` für **UNIX**-Systeme (statisches Binden). Diese Bibliothek befindet sich im Verzeichnis `$RVSPATH/system`. Das Programm `$RVSPATH/system/rvscd` demonstriert die übliche Nutzung der C-CAL-Schnittstelle. Den C-Quell-Code für dieses Programm finden Sie in der Datei `$RVSPATH/samples/s_rvscd.c` und die dazugehörige make-Datei `s_make.cd` im Verzeichnis `$RVSPATH/system`.
- `rvscal` für **OS/400** (Service-Programm). Dieses Programm befindet sich in der Bibliothek `RVS_SYSTEM`.

Dann muss das benutzerdefinierte C-Programm, das diese Bibliothek benutzt, die Header-Datei `rvscal.h` in den Quell-Code einbinden.

Prototyp der Funktion `rvscal`

```
int rvscal(char s_cmd[],
   int *p_l_cmd,
   char *p_c_msglvl,
   char s_msg[],
   int *p_l_msg,
   long *p_cmdid);
```

Hinweis: Die Parameterliste wurde so gestaltet, dass auch Aufrufe durch andere Programmiersprachen möglich sind.

4.1.2.1 Beschreibung der Parameter

`RETURNvalue` (int)

=**0**, wenn `rvscal` erfolgreich war (obwohl der Nachrichtentext abgeschnitten sein kann); von **0** verschieden, wenn ein Fehler auftrat.

Es gibt zwei Arten von Fehlercodes:

- Fehler die vom Kommando-Parser wegen einer ungültigen Kommandosyntax ausgelöst wurden. Die Fehlercodes liegen im Bereich zwischen 1 und 7. Eine genaue Beschreibung finden Sie im "Meldungs- und Return-Code-Handbuch".
- Andere Fehler, die auftraten, während rvs® versuchte, das angegebene Kommando auszuführen. Eine Beschreibung der Fehlerursache wird mit `s_msg` zurückgegeben.

`s_cmd` (char [], input)

Zeiger auf das Character-Array mit dem Kommando-String; die Größe des Array muss mindestens ***p_l_cmd+1** sein.

`p_l_cmd` (int *, input)

Länge des Kommando-Strings; `s_cmd` muss null-terminiert sein, wenn ***p_l_cmd** größer als die Länge des Kommando-Strings ist.

`p_c_msglvl` (char *, output)

gibt das Level der Nachricht zurück; möglich sind **I** (informativ), **W** (Warnung), **E** (error, Fehler), **S** (severe, schwerer Fehler)

`s_msg` (char [], output)

Zeiger auf den Character-String der Mindestlänge ***p_l_msg+1**; der Character-String erhält eine Nachricht von `rvscal`, die Informationen über den Erfolg des Aufrufs von `rvscal` gibt.

`p_l_msg`	(int *, output) Beim Funktionsaufruf ist ***p_l_msg+1** die Größe des Puffers der für den Empfang der Nachricht von `rvscal` bereitgestellt wird. Bei der Ausgabe ist ***p_l_msg** die tatsächliche Länge des Nachrichtentextes. Dieser Wert schließt nicht den Null-Terminator ein, sondern `s_msg` ist null-terminiert.
`p_cmdid`	(long *, output) Wird durch die Funktion `rvscal()` mit dem rvs® Kommandonummer CMDID gefüllt.

Folgende Kommandos werden in `s_cmd []` übergeben:

`START`	Startet die rvs® Utility-Sitzung
`END`	Beendet die Sitzung
`SEND`	Einen Sendeauftrag erzeugen, ändern (anhalten oder freigeben) oder löschen
`PRINT`	Dokumentation über rvs® drucken
`RECEIVER`	einen einzelnen Empfänger aus dem Sendeauftrag löschen
`RESENTR`	einen residenten Empfangseintrag erzeugen, aktualisieren oder löschen
`SENDJOB`	einen Eintrag für den Jobstart nach Sendeversuch erzeugen, aktualisieren oder löschen
`USER`	einen Eintrag in der rvs® Benutzertabelle erzeugen, aktualisieren oder löschen
`ACTIVATE`	eine rvs® Station aktivieren
`MODST`	eine Stationstabelle ändern (eine Stationstabellen-Datei lesen)
`DELST`	einen Eintrag in der Stationstabelle löschen
`LISTPARM`	einen rvs® Parameter auflisten
`SETPARM`	einen rvs® Parameter setzen

Die Kommandonummer **CMDID**, die von rvscal zurückgegeben wird, hat die folgenden Bedeutungen abhängig von dem an rvscal gegebenen Kommando:

Kommando	**Bedeutung der Kommandonummer CMDID**
SEND /CREATE	Nummer des erzeugten Sendeeintrages
SEND /DELETE	Nummer des gelöschten Sendeeintrages[1]
SEND /HOLD	Nummer des angehaltenen Sendeeintrages (siehe Fußnote)
SEND /RELEASE	Nummer des freigegebenen Sendeeintrages (siehe Fußnote)
RESENTR /CREATE	Nummer des erzeugten residenten Empfangseintrages
RESENTR /DELETE	Nummer des gelöschten residenten Empfangseintrages
RESENTR /UPDATE	Nummer des neuen residenten Empfangseintrages
SENDJOB /CREATE	Nummer des erzeugten Eintrages für Jobstart nach Sendeversuch
SENDJOB /DELETE	Nummer des gelöschten Eintrages für Jobstart nach Sendeversuch
SENDJOB /UPDATE	Nummer des neuen Eintrages für Jobstart nach Sendeversuch
RECEIVER /DELETE	Nummer des korrespondierenden Sendeeintrages
LISTPARM	ungleich **0**: ungültiger Parameter
SETPARM	ungleich **0**: ungültiger Parameter
MODST	ungleich **0**: während des Ladens der Stationstabelle aufgetretener Fehler
Jedes andere Kommando	**0**

[1] Wenn der Sendeeintrag durch die angegebenen Parameter nicht eindeutig bestimmt werden kann, wird die Nummer des ersten passenden Kommandos zurückgegeben.

5 Beschreibung der Kommandos

Dieser Abschnitt beschreibt die Syntax der Kommandos sowie den von der C-CAL-Schnittstelle (`rvscal`) und der Kommandozeilen-Schnittstelle (`rvsbat`) benutzten gültigen Kommandosatz.

Die unten stehenden Beispiele benutzen zwei Syntax- Erweiterungen, die nur für die Kommandozeilen-Schnittstelle verfügbar sind:

- Kommentarzeilen beginnen mit einem ***** in der Spalte **1**,
- Kommandos können in der nachfolgenden Zeile weitergeführt werden, indem ein **+** als letztes Zeichen der fortzusetzenden Zeile steht.

5.1 Syntax der Kommandos

Ein Utility-Kommando muss der untenstehenden Syntax folgen:

- Es besteht aus
  - einem Kommando-Verb,
  - einem optionalen Bestimmungswort wie `/CREATE`, `/DELETE`, usw. Das Bestimmungswort kann mit **/** oder **–** beginnen und darf auf einen Buchstaben abgekürzt sein.
  - Werte nicht-wiederholbarer Parameter, angegeben durch <Parametername>=<Parameterwert>. Vor und nach dem Gleichheitszeichen "=" darf kein Leerzeichen stehen.
  - Werte wiederholbarer Parameter (nur Kommando `send`); diese folgen denselben Syntaxregeln wie für nicht-wiederholbare Parameter. Eine Gruppe wiederholbarer Parameter wird in Klammern gesetzt. Es gibt beliebig viele Gruppen wiederholbarer Parameter.
- Kommando-Verb, Bestimmungsworte und Parameternamen können in Groß-, Kleinbuchstaben oder gemischt angegeben sein.
- Parameterwerte werden in Großbuchstaben konvertiert, wenn sie nicht durch einfache oder doppelte Anführungszeichen geschützt sind.
- Wenn ein Parameterwert das schützende Zeichen selbst enthalten soll, muss das schützende Zeichen zweimal angegeben sein. Z.B. sind die folgenden Angaben gleichbedeutend.

  ```
  PARM=' "test string" ' or parm="""test
  string"""
  ```

  Nicht geschützte Parameterwerte können beliebige alphanumerische Zeichen enthalten. Sie können einen Satz spezieller

Zeichen nicht enthalten, z.B. Leerzeichen, einfache oder doppelte Anführungszeichen und Klammern.

- Unterschiedliche Parameterangaben müssen durch mindestens ein Leerzeichen voneinander getrennt sein.

5.2 Das Kommando `START`

Funktion:

- Eine Sitzung mit den rvs® Utilities starten
- rvs® Datenbank öffnen,
- Prüfen, ob der aktuelle Benutzer die rvs® Utilities benutzen darf
- `start /USER` wird implizit durch die C-CAL-Schnittstelle aufgerufen, wenn es nicht explizit aufgerufen wird. Wenn die Kommandozeilen-Schnittstelle die Ausführung startet, wird `start /USER` immer implizit aufgerufen.

Bestimmungsworte:

`/USER` (Standard) startet die Sitzung für den rvs® Benutzer

Parameter:

RVSENV (optional) Name der rvs® Umgebungsdatei

Beispiele:

Umgebungsdatei in (rvs400 Bibliothek) **RVS_NEW/DAT(RVSENV)** benutzen:

```
START /USER RVSENV="RVS_NEW/DAT(RVSENV)"
```

5.3 Das Kommando END

Funktion:

- Beendet die Sitzung mit den rvs® Utilities
- Schließt die rvs® Datenbank
- END muss bei der Verwendung der C-CAL-Schnittstelle aufgerufen werden. Wenn die Kommandozeilen-Schnittstelle die Ausführung startet, wird END immer implizit aufgerufen.

Bestimmungsworte:

keine

Parameter:

keine

Beispiele:

```
END
```

5.4 Das Kommando `SEND`

Funktion:

- Erstellt
- ändert oder
- löscht

einen Sendeauftrag für die Übertragung einer lokalen Datei an einen anderen rvs® Knoten.

Bestimmungsworte:

`/CREATE` oder `/C`	(Standard) erstellt einen Sendeauftrag
`/DELETE` oder `/D`	löscht einen Sendeauftrag
`/HOLD` oder `/H`	setzt den wartenden Sendeauftrag in den Status "angehalten" (held)
`/RELEASE` oder `/R`	Sendeauftrag freigeben, der zuvor in den Status "angehalten" gesetzt wurde

`SEND /CREATE` Parameter:

DSN	(erforderlich) Name der zu sendenden lokalen Datei, hier ist der ganze Pfad der Datei anzugeben.
CODEIN	(optional) Code der lokalen Datei (**A**=ASCII, **E**=EBCDIC); Standard: lokaler Code des Systems.
DISP	(optional) Disposition für die lokale Datei, nachdem der Sendeauftrag erfolgreich ausgeführt ist: **K**=halten (keep, Datei wird nicht gelöscht nach dem Versand, Standard), **D**=löschen (delete, Datei wird gelöscht nach dem Versand).

FORMAT (optional) das für die Übertragung mit O-DETTE benutzte Format:

T=Text (eine Folge von ASCII-Zeichen), **U**= unstrukturiert (binär), **F**=feste Satzlänge, **V**=variable Satzlänge;

Standard: Satzformat der lokalen Datei

(**U** für rvsNT, rvsX und rvs2, **F** für rvs400)

ACCOUNT Berechnungsnummer oder Code.

INITTIME (optional) frühest möglicher Zeitpunkt, zu dem der Sendeauftrag ausgeführt werden kann; möglich sind: **H**=anhalten (hold), **N**=jetzt (now, Standard), oder eine bestimmte Zeit im Format **JJJJ/MM/TT HH:MM** (**Beispiel**: 2004/09/04 10:43).

SERIAL (optional) bei **Y** (=Ja) wird der Sendeauftrag in der von Ihnen eingegebenen Reihenfolge gesendet (siehe auch **LABEL**). Die Datei wird erst gesendet, wenn der vorhergehende Sendeauftrag vollständig erledigt ist.

LABEL (optional) Benutzerkennung (bis zu 20 Zeichen) für die Serialisierung eines vorangehenden Sendeauftrages (bei **SERIAL**=**Y**). Alle Dateien, die in der gleichen Gruppe versendet werden, müssen die gleiche Kennung (**LABEL**) haben.

VFTYP (optional) Textdateien können auch im Format Fest(**F**) oder Variable(**V**) versendet werden, ohne mit dem rvs®-Hilfswerkzeug `rvsut2fv` bearbeitet zu werden.

VFTYP=T (text) bedeutet, dass Ihre Textdatei ohne `rvsut2fv` versendet wird.

Textdatei bedeutet eine Folge von ASCII-Zeichen, wobei bei den einzelnen Satzlängen, der Zeilenwechsel (CR/LF bei MS Windows, LF bei UNIX-Systemen) nicht dazugerechnet wird. Neben dem Parameter **VFTYP** müssen auch die Parameter **MAXRECL** und **FORMAT** gesetzt werden. **Beispiel**: Wenn Sie eine Textdatei im Format **F** mit Satzlänge 80 versenden möchten, muss Ihre Textdatei in jeder Zeile 80 Zeichen lang sein (ohne CR/LF oder LF). Folgende Parameter sind dann wie folgt zu setzen: **VFTYP=T**, **FORMAT=F**, **MAXRECL=80**.

Wenn Sie aber Ihre Textdatei im Format **V** versenden möchten, bedeutet dies, dass Sie eine Textdatei (eine Folge von ASCII-Zeichen) mit unterschiedlichen Zeilenlängen vorbereitet haben (z.B. der längste Satz ist 120 Zeichen lang) und die notwendigen Parameter sind dann wie folgt zu belegen: **VFTYP=T**, **FORMAT=V**, **MAXRECL=120**. Als **MAXRECL** ist die maximale Länge eines Satzes bis zum Zeilenwechsel gedacht (ohne CR/LF oder LF). Siehe auch Beispiele am Ende des Abschnittes.

VFTYP=V (variabel) bedeutet, dass die Datei vor dem Versand mit `rvsut2fv` behandelt wurde. Dann ist nur das Format (**FORMAT**) der zu sendenden Datei anzugeben, ohne den Parameter **MAXRECL**. Siehe Beispiele am Ende des Abschnittes.

MAXRECL (optional) maximale Satzlänge für Dateien im Format **T**, **F** oder **V**; dieser Parameter wird nur benötigt, wenn die Dateien nicht mit dem rvs®-Hilfswerkzeug `rvsut2fv` davor konvertiert wurden. Siehe auch Parameter **VFTYP**.

SIDORIG ID der virtuellen Station beim Senden; der Wert dieses Parameters wird beim Senden in die SFID (Start File ID) eingefügt.

Die folgende Gruppe der Sendeparameter sind innerhalb der runden Klammern () anzugeben:

SID (erforderlich) Stations-ID des Empfängers

UID (optional) Benutzer-ID des Empfängers; wenn kein Wert oder einer leerer String angegeben ist, ist dies des System auf dem fernen Station

COMPRESSION (optional) bei **Y** (=Ja) wird die Datei vor dem Senden komprimiert. Dieser Parameter muss beim Aufruf nach dem **SID** Parameter in Klammern angegeben werden; z.B.

```
SEND /C DSN=/home/test/test11.txt
(SID=RTZ COMPRESSION=Y)
```

ENCRYPTION (optional) bei **Y** (=Ja) wird die Datei vor dem Senden verschlüsselt. Dieser Parameter muss beim Aufruf nach dem **SID** Parameter in Klammern angegeben werden; z.B.

```
SEND /C DSN=/home/test/test11.txt
(SID=RTZ COMPRESSION=Y ENCRYP-
TION=Y)
```

DSNNEW (optional) Für die Übertragung benutzter virtueller Dateiname (VDSN).

Bei Übertragungen an einen MVS-Hostrechner, muss dies ein gültiger MVS-Dateiname sein (den RACF-Regeln entsprechen).

Standard: VDSN wird aus dem lokalen Dateinamen gebildet. Die maximale Länge von VDSN ist nach ODETTE 26 Zeichen.

CODEOUT (optional) Gewünschter Code (**A**=ASCII, **E**=EBCDIC) der Datei beim Empfänger. Z.B.

```
send /c dsn=C:\test22.dat co-
dein=a format=v(sid=rtt codeout=e
dsnnew=FIX0GBE.TEXT)
```

CODETABLE (optional) Definiert die Codetabelle, die zur Codeumwandlung (siehe auch Abschnitt über Codeumwandlung im Benutzerhanbuch) verwendet werden soll. Hier ist der ganze Pfad der Codetabelle-Datei anzugeben.
Beispiel:

```
send /c
dsn=/home/send/test22.dat co-
dein=a format=v (sid=rtt code-
out=e codetable=/home/tables
/rtcusrdat dsnnew=FIX0GBE.TEXT)
```

TSTAMP (optional) Gibt an, ob die Datei bei dem Empfänger einen Zeitstempel erhalten soll.

Zurzeit kann diese Anforderung an einen oder von einem MVS-Hostrechner nicht übergeben werden.

SEND /DELETE, /HOLD, /RELEASE Parameter:

DSN (optional) Name der zu sendenden lokalen Datei

SID (optional) Stations-ID des Empfängers

UID (optional) Benutzer-ID des Empfängers; im Standard ein leerer String, d.h. das ferne System

CMDID (optional) Eindeutige Kommandonummer des Sendeauftrages.

Der zu bearbeitende Sendeauftrag kann entweder durch **DSN**, **SID**, und **UID** oder durch die eindeutige Kommandonummer **CMDID** des Sendeauftrags identifiziert werden.

Wenn kein Parameter angegeben wurde oder mehr als ein Sendeauftrag mit den angegebenen Parametern übereinstimmt, ändert `rvscal` keinen Sendeauftrag und gibt einen Fehler zurück.

Die folgenden Beispiele zeigen, wie Kommandos in einer Datei angegeben werden können, die als Eingabe-Datei für `rvsbat` benutzt wird. Der Gebrauch von Fortsetzungszeilen (die vorhergehende Zeile endet mit **+**) und Kommentaren (* in Spalte 1) wird ebenso gezeigt.

Beispiele:

```
SEND /C DSN=C:/RVS/LPDBI.C +
SERIAL=n LABEL=l1 inittime=NOW +
CODEIN=A FORMAT=T DISP=d +
(SID=st1 UID=user1 CODEOUT=e TSTAMP=n
DSNNEW=dsnnew1)

SEND  /C DSN=/HOMe/LPDBI.C +
SERIAL=y LABEL=l1 inittime=HOLD +
FORMAT=U DISP=k (SID=st1)

SEND  /C DSN=rpu.c +
SERIAL=y LABEL=l1 inittime='1991/07/01 10:35' +
FORMAT=U (SID=st1)

*------ serialize on data set (without specifying
* the full data set name)

SEND  /C DSN=lpdbi.c SERIAL=y FORMAT=T (SID=st1)

*------ serialize again on data set (and default
* for FORMAT)
*
SEND  /C DSN=lpdbi.c SERIAL=y (SID=st1)


*------ delete SEND request (unique)
SEND /D DSN=/home/rpu.c (SID=st1)
```

```
* --- Textdatei zum Host senden ohne rvsut2fv
send /c dsn=C:\test.text codein=a format=f
vftyp=t maxrecl=80(sid=rtt codeout=e
dsnnew=FIX0GBE.TEXT)

* --- Textdatei, die mit rvsut2fv konvertiert
* --- wurde zum Host senden
send /c dsn=C:\test22.text codein=a format=v
(sid=rtt codeout=e dsnnew=FIX0GBE.TEXT)
```

5.5 Das Kommando `RESENTR`

Funktion:

Erstellen, Überarbeiten und Löschen von residenten Empfangseinträgen.

Bestimmungsworte:

`/CREATE` oder `/C`	(Standard) erstellt einen residenten Empfangseintrag
`/UPDATE` oder `/U`	überarbeitet einen residenten Empfangseintrag
`/DELETE` oder `/D`	löscht einen residenten Empfangseintrag

Parameter:

DSN	(erforderlich) Virtueller Name der eintreffenden Datei (VDSN, darf max. 26 Zeichen lang sein).
SID	(erforderlich) Stations-ID des Senders.
ACCOUNT	(optional) Berechnungsnummer oder Code
COMMENT	(optional) Kommentar zur Beschreibung des residenten Empfangseintrages (bis zu 50 Zeichen lang); Standard: leerer String
DSNNEW	(optional) der neue Dateiname auf dem lokalen System. Hier ist der ganze Pfad der neuen Datei anzugeben. Standard: Der lokale Dateiname wird aus dem virtuellen Dateinamen gebildet.
JOB	Name einer Kommandozeilen-Datei die nach der Zustellung der Datei gestartet wird[1] (.bat unter Windows-Systemen; .sh für UNIX-Systeme).

[1] Die Art des geforderten Jobs hängt vom Betriebssystem ab. Z.B. wird unter OS/2 eine CMD-Datei als Hintergrundprozeß ausgeführt, während unter OS/400 rvs® ein Job mit Hilfe von **SBMDBJOB** startet.

Diese Kommandozeilen-Datei darf Ersetzungszeichen enthalten. rvs® ersetzt sie, bevor es den Job zur Ausführung an das Betriebssystem übergibt.

- ?**DSN**?: Name der lokalen Datei, in der die empfangene Information gespeichert wurde.
- ?**VDSN**?: Virtueller Dateiname, unter welchem die Datei übertragen wurde.
- ?**DTAVAIL**?: Datum, zu dem die Datei zum Senden verfügbar war.
- ?**FORMAT**?: Satzformat der empfangenen Datei.
  - **F** feste Satzlänge
  - **V** variable Satzlänge
  - **T** Textdatei (ASCII-Zeichenfolge)
  - **U** unstrukturierte Datei (binär)
- ?**MAXRECL**?: Die Bedeutung dieses Feldes hängt vom Satzformat der empfangenen Datei ab.
  - **F** Format: Länge jedes Satzes.
  - **V** Format: Maximale Länge, den ein Satz haben kann.
  - **T** und **U** Format: Immer **0** (null)
- ?**BYTES**?: Zahl der übertragenen Bytes
- ?**RECORDS**?: Zahl der übertragenen Sätze für Dateien im **F** und **V** Format; immer **0** (null) für Dateien im **T** und **U** Format
- ?**DTRCV**?: Datum, an dem die Datei dem lokalen Benutzer zugestellt wurde
- ?**LUID**?: ID des (lokalen) Empfängers
- ?**UID**?: ID des Senders
- ?**SID**?: Stations-ID des Empfängers
- **?SIDDEST?** StationsID der virtuellen Empfangsstation
- ?**CNQS**? Kommandonummer der Quittungs-Sendung (EERP) für die empfangene Datei
- ?**DSNTEMP**?: Name der temporären Datei (Diese Datei kann am Ende des Sendeauftrages mit dem Befehl

– `DELETE ?DSNTEMP?` (**NT**, **XP**, **2000**)

– `rm ?DSNTEMP?` **(UNIX)**

– `DLTF ?DSNTEMP?` (**OS/400**)

gelöscht werden.)

DISP (optional) Bestimmt die Behandlung der Datei, wenn die Bearbeitung abgeschlossen ist.

- **K** hält die Datei
- **D** löscht die Datei

Standard: **K**

REPLACE (optional) Wenn **DISP**=**K** (halten) eingestellt ist, legt **REPLACE** die Aktionen fest, die rvs® durchführt, nach dem Empfang einer Datei, die im Namen mit einer lokalen Datei übereinstimmt.

R Vorhandene Datei ersetzten

N Neuen eindeutigen Dateinamen verwenden

I Eintreffen der Datei ignorieren

Standard: **N**

TSTAMP (optional) Kann **Y**=Ja oder **N**=Nein sein; teilt rvs® mit, ob der Dateiname zur eindeutigen Kennzeichnung mit Zeitstempel versehen werden soll, wenn die Datei eingetroffen ist..

Standard: **N**

CODETRANS (optional) Gibt an, ob die empfangene Datei mit rvs-internen Codeumwandlungstabellen in ASCII oder EBCDIC konvertiert werden soll, oder ob zu Codeumwandlung eine eigene Codetabelle benutzt werden soll (siehe auch Kapitel über Codeumwandlung im rvs® Benutzerhanbuch).

A für die Codeumwandlung nach ASCII

E für die Codeumwandlung nach EBCDIC

T für die Codeumwandlung mit Codetabelle

z.B. `resentr /c dsn="<empfangene ASCII Datei>" codetrans=e sid="<Sender>"`

CODETABLE (optional) Definiert die eigene Codeumwandlungstabelle (siehe auch Abschnitt über Codeumwandlung im Benutzerhanbuch), die verwendet werden soll.

z.B. `resentr /c dsn="<empfangene EBCDIC Datei>" codetrans=t codetable ="<User Code Tabelle, z.B. /home/rvs/arcdir/rtcusrdat>"`

```
sid="<Sender>"
```

VFTYP (optional) Sie können hier bestimmen, ob die empfangene Datei als Textdatei mit Zeilenwechsel (CR/LF bei MS Windows, LF bei Unix-Systemen) nach jedem Satz abgelegt werden soll. Diese Angabe gilt nur für Dateien, die im Format **Fixed** oder **Variable** empfangen werden.

VFTYP=T (text) bedeutet, dass die empfangene Datei als Textdatei mit Zeilenwechsel gespeichert werden soll.

VFTYP=V (variabel) bedeutet, dass die empfangene Datei im Format **Fixed** oder **Variable** ohne Umwandlung in Textdateien im rvs® internen Format gespeichert wird.

VFTYP=S (Sinix) bedeutet, dass die empfangene Datei im Sinix-Format gespeichert wird.

Beispiele:

```
*
*---------- use all parameters
*
RESENTR /C DSN=incoming1  SID=st2 +
 DSNNEW=/home/local.dsn REPLACE=n DISP=k +
 JOB=/home/rvs/bin/rcv.sh COMMENT='This is a test
RE'
*
*---------- use defaults
*
RESENTR /C DSN=incoming2  SID=st2 +
 REPLACE=i    DISP=d
*
*---------- no UID, DISP
*
RESENTR /C DSN=incoming3  SID=st2 +
 REPLACE=r
*
*---------- delete RESENTR
*
RESENTR /D DSN=incoming3  SID=st2
*
*---------- update RESENTR
*
RESENTR /U DSN=incoming2  SID=st2 +
REPLACE=n    JOB=/home/rvs/bin/rcv.sh
```

5.6 Kommando `SENDJOB`

Funktion:

Erstellen, Überarbeiten und Löschen von Einträgen für Jobstart nach Sendeversuch.

Bestimmungsworte:

`/CREATE`	(Standard) erstellt einen Eintrag für Jobstart nach Sendeversuch
`/UPDATE`	überarbeitet einen residenter Empfangseintrag
`/DELETE`	löscht einen residenten Empfangseintrag

Parameter:

VDSN (erforderlich) Virtueller Name der gesendeten Datei.

SID (erforderlich) Stations-ID des Empfängers

ATTEMPTS (optional) Zahl der Sendeversuche; legt fest, bei welcher Bedingung der Job starten soll

Wert **0**: Der Job startet bei erfolgreicher Übertragung

Wert >**0**: Der Job startet, wenn die angegebene Zahl fehlgeschlagener Sendeversuche erreicht ist

Standard: **0**

JOB (optional) Name des Jobs, der nach erfolgreicher Übertragung oder erfolglosen Sendeversuch gestartet werden soll. Diese Kommandozeilen-Datei (der Job) darf Ersetzungszeichen enthalten. rvs® ersetzt sie, bevor es den Job zur Ausführung an das Betriebssystem übergibt:

- ?**DSN**?: Name der lokalen Datei, die gesendet wurde. Bei **EERP** gibt es keinen lokalen Dateinamen. Der Wert von ?**DSN**? hat die Form **QS** (**SIDORIG** - **SIDDEST**) mit der Bedeutung:

SIDORIG Stations-ID des Senders
SIDDEST Stations-ID des Empfängers

- ?**VDSN**?: virtueller Dateiname unter dem die Datei übertragen wurde.
- ?**DTAVAIL**?: Datum, an dem die Datei für das Senden verfügbar war.
- ?**FORMAT**?: Satzformat der gesendeten Datei
  - **F** fest
  - **V** variabel
  - **T** Text
  - **U** unstrukturiert
- ?**BYTES**?: Zahl der übertragenen Bytes
- ?**RECORDS**?: Zahl der im Format **F** oder **V** übertragenen Sätze
- ?**DTRCV**?: Datum, an dem die Datei dem lokalen Benutzer zugestellt wurde.
- ?**LABEL**?: Zeichenfolge, wenn das Sendekommando ein **LABEL** Parameter enthielt.. Kann zur Identifikation des Sendekommandos benutzt werden.
- ?**SECN**?: Kommandonummer des Sendekommandos (**CN** des **SE**). Kann zur Identifikation des Sendekommandos benutzt werden.
- ?**SKCN**?: Nummer des Sendekommandos
- ?**UID**?: Benutzer-ID des Senders; bei **EERP** ist der Wert immer "!-**QS**-!".
- ?**SID**?: Stations-ID des Empfängers
- ?**SIDORIG**? Stations-ID der virtuellen Senderstation.
- ?**SENDATT**? Anzahl der Sendeversuche, nach denen der Job gestartet werden soll.
- ?**DSNTEMP**?: Name der temporären Datei (Diese Datei kann am Ende des Sendeauftrages mit dem Befehl
  - `DELETE ?DSNTEMP?` (**NT**)
  - `rm ?DSNTEMP?` **(UNIX)**
  - `DLTF ?DSNTEMP?` (**OS/400**) gelöscht werden.)

COMMENT (optional) Ein Kurztext zur Beschreibung des Zwecks des Jobstarteintrages (bis zu 50 Zeichen lang); Standard: leerer String

Beispiele:

```
*
*---------- use all parameters
*
SENDJOB /C VDSN=sending1  SID=st2 ATTEMPTS=1 +
 JOB=/home/rvs/bin/send-fail1.sh COMMENT='This is
a test JS'
*
*---------- use defaults
*
SENDJOB /C VDSN=sending2  SID=st2
JOB=/home/rvs/bin/snd.sh
*
*---------- Job should start after data have been
transmitted successfully
*
SENDJOB /C VDSN=sending3  SID=st2
JOB=/home/rvs/bin/snd.sh
*
*---------- delete SENDJOB
*
SENDJOB /D VDSN=sending3  SID=st2
*
*---------- update SENDJOB
*
SENDJOB /U DSN=/home/sending2  SID=st2
JOB=/home/rvs/bin/sendok.sh
```

5.7 Das Kommando USER

Funktion:

- rvs® Benutzertabelle ändern
- Benutzer einstellen
- Dialogsprache für Benutzer festlegen
- Privilegien für die Benutzer festlegen

Bestimmungsworte:

/CREATE	erstellt einen Eintrag in die rvs® Benutzertabelle
/DELETE	löscht einen Eintrag in die rvs® Benutzertabelle
/UPDATE	überarbeitet einen Eintrag in die rvs® Benutzertabelle

Wenn kein Bestimmungswort angegeben ist, wird der Eintrag erstellt oder überarbeitet, je nachdem, ob der Eintrag schon besteht oder nicht.

Parameter:

UID	(optional) Benutzer-ID, die den zu ändernden Eintrag identifiziert; Sie können nur als privilegierter Benutzer etwas anderes als die aktuelle Benutzer-ID angeben. Standard: current user
NAME	(optional) neuer Benutzername
LANGUAGE	(optional) Neue Sprache angeben (z.B. **E**=Englisch, **D**=Deutsch)
PRIV	(optional) neues Benutzerprivileg; (**U**=Benutzer, **O**=Operator, **A**=Administrator)

Beispiele:

```
*---------- create new user (default=own UID)
*
USER /C
*
*---------- create new user
*
USER /C UID=newuser LANGUAGE=d NAME=x PRIV=O
```

```
USER /C UID=extrauser
*
*---------- update existing user
*
USER /U UID=newuser LANGUAGE=e NAME=y PRIV=U
*
*---------- delete user
*
USER /D UID=newuser
```

5.8 Das Kommando ACTIVATE

Funktion:

- Aktiviert eine Partnerstation
- rvs® startet einen Sendeprozess, der die Verbindung zur Partnerstation herstellt. Alle Dateien in Wartestellung werden übertragen. Danach wird die Verbindung wieder getrennt.

Parameter:

SID (erforderlich) Stations-ID des Partners

Beispiele:

```
*
*---------- activate station ABC
*     (connect and send or receive queued data
sets)
*
ACTIVATE SID=ABC
```

5.9 Das Kommando `MODST`

Funktion:

Stationstabelle ändern.

Achtung: Dieses Kommando überschreibt alte Datenbankeinträge; andere vorhandene Einträge werden nicht gelöscht () nur **DELST** löscht einen Eintrag).

Parameter:

DSN (erforderlich) Name der Datei, die die Stationstabelle enthält. Der Dateiname kann eine einfache Eingabedatei oder ein Verzeichnis mit mehreren darin enthaltenen Eingabedateien sein.

Beispiele:

```
*
*---------- modify station table:
*     (open the file (here: UNIX file name),
*     read contents and put it into database):
*
MODST DSN="/home/rvs/init/new_rdstat.dat"
```

5.10 Das Kommando DELST

Funktion:

Stationstabelle löschen.

Parameter:

SID Stations-ID der zu löschenden Station

Beispiele:

```
*
*---------- delete entry ABC in database:
*
DELST SID=ABC
```

5.11 Das Kommando `LISTPARM`

Funktion:

- Gibt den Wert eines rvs® Parameters aus.
- Im Kommandozeilen-Modus erscheint die Meldung "I: Wert".
- In C-Programmen wird der Wert als String im `rvscal()`-Ausgabeparameter **S_MSG** übergeben.

Parameter:

parameter Name des rvs® Parameters.

Beispiele:

```
*
*---------- show the value of the parameter SLEEP
*
LISTPARM SLEEP
```

5.12 Das Kommando SETPARM

Funktion:

Setzt den Wert eines rvs® Parameters.

Parameter:

parameter Name des rvs® Parameters.

Beispiele:

```
*
*---------- set the value of the parameter SLEEP
*
SETPARM SLEEP=2
```

6 Arbeiten mit C-Funktionen

Die folgenden Abschnitte beschreiben, wie Sie die C-Sprachen-Funktionen benutzen, die Sie in ein Anwendungsprogramm einbinden können, um die rvs® Kommandos auszuführen. Bitte überprüfen Sie bei rvs® Updates `rvscal.h` nach letzten Änderungen von Strukturen und Funktionsprototypen.

6.1 Senden und Empfangen mit der C-CAL-Schnittstelle

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Sendeeinträgen mit der C-CAL-Schnittstelle erforderlich sind. Für alle Funktionen gelten folgende Typ-Definitionen und die zugehörigen Prototypen.

6.1.1 Typ-Definitionen

```
typedef struct {
 SINT i_error;
 SINT i_type;
 char s_uid [RVSCAL_L_USID];
 char s_jobid [RVSCAL_L_JOBID];
 char sid_neighb [RVSCAL_L_STATID];
 char sid_dest [RVSCAL_L_STATID]; /* destination
SID */
 char sid_sender [RVSCAL_L_STATID];
 char s_vdsn [RVSCAL_L_VDSN];
 char dt_created [RVSCAL_L_DT]; /* job creation
date and time */
 char dt_avail [RVSCAL_L_DT];
 char dt_sched [RVSCAL_L_DT];
 char dt_begin [RVSCAL_L_DT];
 char dt_end [RVSCAL_L_DT];
 char dt_done [RVSCAL_L_DT];
 char dt_received [RVSCAL_L_DT];
 long int cnt_sendatt; /* number of send attempts
*/
 long int cnt_record;
 long int cnt_byte; /*number of already sent
bytes*/
 long int cnt_maxrecl;
 long int cnt_apsize;
 long int cnt_lenvm;
 long int cn_se;
 long int cn_sk;
```

```
 long int cn_ie;
 long int cn_iz;
 long int cn_re;
 char status_et [RVSCAL_L_KS];
 char status_se [RVSCAL_L_KS];
 char status_sk [RVSCAL_L_KS] ;/* state of send
cmd (-|a|i|f|p|e|d|s) */
 char status_ie [RVSCAL_L_KS];
 char status_iz [RVSCAL_L_KS];
 char dsn_local [RVSCAL_L_DSN];
 char s_recfm [RVSCAL_L_C1];
 char s_ftype [RVSCAL_L_C1];
 char s_code [RVSCAL_L_C1];
 char s_codein [RVSCAL_L_C1];
 char s_codeout [RVSCAL_L_C1];
 char s_disp [RVSCAL_L_C1];
 char s_label [RVSCAL_L_SEUSRLABEL]; /* user de-
fined label */
 SINT flg_tstamp;
} INFO_SK ;


/*SetSendEntry Commands: */
#define SET_HOLD    1
#define SET_RELEASE  2
#define SET_DELETE   3


#define TRANSMISSION_SEND 1
#define TRANSMISSION_RECV 2


#define CN_START     0
```

6.1.2 Nächsten Sendeauftrag aus der Datenbank holen

Prototyp `rvsGetNextSend`:

```
PROCDEF int PROCKEYW rvsGetNextSend(long
prev_send_cn, INFO_SK *info);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn rvs® einen nächsten Sendeeintrag gefunden hat.

=**RVSCAL_END_FETCH**, wenn keine Sendeeinträge größer als **PREV_SEND_CN** vorhanden sind.

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

prev_send_cn (int, input)

Kommandonummer des vorherigen Sendeeintrages

info (struct **INFO_SK ***, output)

Struktur mit Informationen über den nächsten Eintrag mit einer Kommandonummer größer als **prev_send_cn**

REMARKS `rvsGetNextSend` sucht nach dem nächsten Eintrag mit einem Wert größer als **prev_send_cn**. Wenn die Funktion das erste Mal aufgerufen wird, muß **prev_send_cn CN_START** sein**. CN_START** löste `rvsGetNextSend` aus, um Einträge von der rvs® Datenbank zu lesen und sie in einer internen Liste zu speichern. Jeder Aufruf von `rvsGetNextSend` mit **prev_send_cn**=**CN_START** aktualisiert diese Liste.

6.1.3 Einen Sendeauftrag aus der Datenbank holen

Prototyp `rvsGetSendEntry`:

```
PROCDEF int PROCKEYW rvsGetSendEntry(const long
send_cn, INFO_SK *info);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn wir die Informationen über das Sendekommando gefunden haben

=**RVSCAL_END_FETCH**, wenn es keinen Sendeeintrag mit der angegebenen Nummer gibt

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler auftrat

send_cn (int, input)

Kommandonummer eines Sendeeintrages

info (struct **INFO_SK ***, output)

Struktur mit Informationen über den Sendeeintrag

6.1.4 Debug-Modus einschalten

Prototyp `rvsSetDebugMode`:

```
PROCDEF int PROCKEYW rvsSetDebugMode(int mode);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn der Debug-Modus eingeschaltet ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

MODE (int, input)

Modustyp

6.1.5 Status von SE ändern

Prototyp rvsSetSendEntry:

```
PROCDEF int PROCKEYW rvsSetSendEntry(long int
cn_se, int SetCmd, char *szSID, char *s_msg);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn kein Fehler aufgetreten ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler auftrat

cn_se (long int, input)

Kommandonummer des SE (zurückgegeben von CreateSendEntry)

SetCmd (int, input)

Kommandotyp

SET_HOLD : Diesen Sendeauftrag anhalten

SET_RELEASE : Diesen Sendeauftrag freigeben

SET_DELETE : Diesen Sendeauftrag löschen

szSID (char *, input)

Stations-ID des Empfängers

s_msg (char *, output)

Fehlermeldung bei einem Fehler

6.1.6 Nächsten Informationseintrag holen

Prototyp `rvsGetNextIE`:

```
PROCDEF int PROCKEYW rvsGetNextIE(long int
prev_ie_cn, INFO_SK *p_info);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn kein Fehler aufgetreten ist.

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

prev_ie_cn (long int, input)

Kommandonummer des vorhergehenden

p_info (Struktur **INFO_SK** *, output)

Datenstruktur mit Informationen über den Eintrag

6.1.7 Eine Datei senden

Prototyp `rvsCreateSendEntry`:

```
PROCDEF int PROCKEYW rvsCreateSendEntry(
  char *dsn,
  char *disp,
  char *format,
  char *codein,
  char *inittime,
  char *serial,
  char *label,
  char *tstamp,
  char *sid,
  char *dsnnew,
char *codeout,
  char *s_msg);
```

Beschreibung der Parameter

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, wenn die Funktion erfolgreich war
	=**RVSCAL_INTERNAL_ERROR**, wenn ein Fehler aufgetreten ist
dsn	(char *, input)
	Dateiname der lokalen Datei (Details siehe rvs® Benutzerhandbuch)
disp	(char *, input)
	Disposition (**K**=halten, **D**=nach Übertragung löschen)
format	(char *, input)
	Dateiformat (**T**/**F**/**V**/**U**)
codein	(char *, input)
	Code der Eingabedatei (**A**=ASCII, **E**=EBCDIC)
inittime	(char *, input)
	Zeitpunkt des frühest möglichen Sendeversuchs (**YY**/**MM**/**DD HH**:**MM**:**SS**)
serial	(char *, input)
	Serialisations-Flag (**Y** oder **N**)

label	(char *, input)
	Dateikennung (für Serialisation)
tstamp	(char *, input)
	Zeitstempel
sid	(char *, input)
	Stations-ID des Empfängers
dsnnew	(char *, input)
	Virtueller Dateiname (**OFTP**)
codeout	(char *, input)
	Ausgangs-Code (**A**=ASCII, **E**=EBCDIC)
s_msg	(char *, output)
	Fehlermeldung bei einem Fehler

6.1.8 Sendeeintrag erstellen

Prototyp `rvsCreateSendEntryCmd`:

```
PROCDEF int PROCKEYW rvsCreateSendEntryCmd(
  char *dsn,
char *disp,
  char *format,
  char *codein,
  char *inittime,
  char *serial,
  char *label,
  char *tstamp,
  char *sid,
  char *dsnnew,
  char *codeout,
  char *s_msg);
```

Beschreibung der Parameter

FUNCTIONVALUE (int)

=Kommandonummer des Sendeeintrages, wenn der Eintrag erfolgreich war

=**RVSCAL_ERROR_CREATESEND**, wenn ein Fehler aufgetreten ist

6.2 Administration mit der C-CAL-Schnittstelle

Dieses Kapitel beschreibt die Funktionen zur Verwaltung von

- Stationseinträgen
- rvs® Parametern
- rvs® Operator-Kommandos
- residenten Empfangseinträge
- Einträgen für Jobstart nach Sendeversuch
- Benutzereinträgen
- Datenbank Funktionen

Für diese Funktionen steht eine Typ-Definition sowie die zugehörigen Prototypen und die Rückgabewerte zur Verfügung.

6.2.1 Funktionen zur Verwaltung von Stationstabellen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Stationstabelleneinträgen erforderlich sind.

6.2.1.1 Typ-Definitionen

```
typedef struct {
 char      netid[RVSCAL_L_NETID];
 char      statname[RVSCAL_L_STATNAME];
 char      phone[RVSCAL_L_PHONE];
} INFO_ST;


typedef struct {
 char      ftp[RVSCAL_L_C1];
 char      protocol[RVSCAL_L_C1];
 char      autodial[RVSCAL_L_C1];
 SINT      pr_nk;
 SINT      flg_suspnd;
} INFO_NK;


typedef struct {
 char      sidneighb[RVSCAL_L_STATID];
 SINT      pr_rt;
} INFO_RT;


typedef struct {
 char      odetteid[RVSCAL_L_ODETTEID];
 char      pswfrom[RVSCAL_L_OPSW];
 char      pswto[RVSCAL_L_OPSW];
 long      i_sendblocks;
 long      i_recvblocks;
 long      i_ocreval;
 long      i_oexbuf;
 char      codein[RVSCAL_L_C1];
 char      codeout[RVSCAL_L_C1];
 char      userfield[RVSCAL_L_OFTP_USERD];
 char      eerp_in[RVSCAL_L_OFTP_EERP];
 char      eerp_out[RVSCAL_L_OFTP_EERP];
 char      vdsnchar[RVSCAL_L_OFTP_EERP];
 char      retry[RVSCAL_L_DT];
} INFO_OP;


typedef struct {
 char      netid_lu[RVSCAL_L_LU62_NETID];
 char      luname[RVSCAL_L_LU62_LUNAME];
 char      tpname[RVSCAL_L_LU62_TPNAME];
 char      userid[RVSCAL_L_LU62_UID];
```

```
 char      password[RVSCAL_L_LU62_PASSW];
 char      profile[RVSCAL_L_LU62_PROF];
 char      mode[RVSCAL_L_LU62_MODE];
 SINT      i_security;
 SINT      flg_sync;
 SINT      flg_conv;
} INFO_LU;


typedef struct {
 char      alias[RVSCAL_L_X25_ALIAS];
 char      recv_alias[RVSCAL_L_X25_ALIAS];
 long      cntn;
 char      xaddr[RVSCAL_L_X25_ADDR];
 char      subaddr[RVSCAL_L_X25_SUBADDR];
 char      timeout[RVSCAL_L_X25_TIMEOUT];
 char      isdnno[RVSCAL_L_X25_ISDNNO];
 char      link[RVSCAL_L_X25_LINK];
 char      fac[RVSCAL_L_X25_FAC];
 SINT      flgdbit;
 char      cug[RVSCAL_L_X25_CUG];
 SINT      flgreqrev;
 SINT      flgaccrev;
 SINT      flgfastsel;
 char      usrdata[RVSCAL_L_X25_USDTA];
 char      vc[RVSCAL_L_C1];
 long      cntsessions;
} INFO_XP;


typedef struct {
 char      protocol[RVSCAL_L_C1];
 long      cntn;
 char      inaddr[RVSCAL_L_TCPIP_ADDR];
 SINT      i_port;
 SINT      i_max_in;
 SINT      i_max_out;
 char      security[RVSCAL_L_TCPIP_SEC];
} INFO_TC;


typedef struct {
 char      lx_name[RVSCAL_L_LX_NAME];
 long      lx_len;
 char      lx_val[RVSCAL_L_LX_VALUE];
} INFO_LX;


typedef struct {
 char      sid[RVSCAL_L_STATID];
 int      flg_st;
 int      flg_nk;
 int      flg_rt;
 int      flg_op;
 int      flg_lu;
 int      flg_xp;
```

```
 int      flg_tc;
 int      flg_lx;
 INFO_ST    st;
 INFO_NK    nk;
 INFO_RT    rt;
 INFO_OP    op;
 INFO_LU    lu;
 INFO_XP    xp;
 INFO_TC    tc;
 INFO_LX    lx;
} INFO_STATION;
```

6.2.1.2 Nächsten Stationseintrag aus der Datenbank holen

Prototyp `rvsGetNextStation`:

```
PROCDEF int PROCKEYW rvsGetNextStation(char
*SIDpre, char * SID);
```

Beschreibung der Parameter

SIDPRE (char *, input)

SID der vorherigen Station

SID (char *, output)

SID der nächsten in **ST** (Stationstabelle) gefundenen Station

6.2.1.3 Stationseintrag in der Datenbank aktualisieren

Prototyp `rvsUpdateStation`:

```
PROCDEF int PROCKEYW rvsUpdateStation( IN-
FO_STATION *info);
```

Beschreibung der Parameter

info (**INFO_STATION** *, input)
Struktur mit Informationen über den Stationseintrag

6.2.1.4 Stationseintrag aus der Datenbank holen

Prototyp `rvsGetStation`:

```
PROCDEF int PROCKEYW rvsGetStation( char
*psz_SID, INFO_STATION *info);
```

Beschreibung der Parameter

psz_SID (char *, input)
SID der Station
info (**INFO_STATION** *, output)
Struktur mit Informationen über den Stationseintrag

6.2.1.5 Stationseintrag aus der Datenbank löschen

Prototyp `rvsDeleteStation`:

```
PROCDEF int PROCKEYW rvsDeleteStation( char
*psz_SID);
```

Beschreibung der Parameter

psz_SID (char *, input)
SID des Stationseintrages

6.2.1.6 Alle unterbrochenen Kommandos wieder aufnehmen

Prototyp `rvsFreeSuspendedCommands`:

```
PROCDEF int PROCKEYW rvsFreeSuspendedCommands(
void);
```

Beschreibung der Parameter

keine Parameter vorhanden

6.2.1.7 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist.

=**RVSCAL_END_FETCH**, wenn es keine Station mit dieser **SID** oder keinen Eintrag größer als **SIDPRE** gibt

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet werden konnte

=**RVSCAL_NEITHER_X_NOR_ISDN**, wenn weder eine X25-Adresse noch eine ISDN-Adresse angegeben wurde

=**RVSCAL_NEIGHBOR_STATION**, wenn bei "delete” Routing-Verbindungen zu dieser Station bestehen

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.2 Funktionen zur Verwaltung von rvs® Parameter

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Parametern erforderlich sind.

6.2.2.1 Typ-Definitionen

```
  typedef struct{
char  s_parm[RVSCAL_L_PARM_NAME];
  SINT  i_type ;
  long  len ;
  char  s_val[RVSCAL_L_PARM_VAL] ;
 } PARM_STRUCT ;
```

6.2.2.2 Parameterwerte aus der Datenbank holen

Prototyp `rvsGetParm`:

```
PROCDEF int PROCKEYW rvsGetParm( char *parm,
PARM_STRUCT *stparm);
```

Beschreibung der Parameter

parm	(char *, input)
	Parametername
stparm	(**PARM_STRUCT** *, output)
	Struktur mit Informationen über den Parametereintrag

6.2.2.3 Nächsten Parameter aus der Datenbank holen

Prototyp `rvsGetNextParm`:

```
PROCDEF int PROCKEYW rvsGetNextParm( char *parm,
PARM_STRUCT *stparm);
```

Beschreibung der Parameter

parm (char *, input)

Vorheriger Parameter

stparm (**PARM_STRUCT** *, output)

Struktur mit Informationen über den nächsten Parametereintrag

6.2.2.4 Parameterwert in die Datenbank schreiben

Prototyp `rvsWriteParm`:

```
PROCDEF int PROCKEYW rvsWriteParm( char *parm,
PARM_STRUCT *stparm);
```

Beschreibung der Parameter

parm (char *, input)

Parametername

stparm (**PARM_STRUCT** *, input)

Struktur mit Informationen über den Parametereintrag

6.2.2.5 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.3 Funktionen zur Verwaltung von rvs® Operator-Kommandos

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Operator-Kommandos erforderlich sind.

6.2.3.1 Operator-Kommando in die Datenbank schreiben

Prototyp `rvsStoreOK`:

```
PROCDEF int PROCKEYW rvsStoreOK( char *command);
```

Beschreibung der Parameter

command (char *, input)
Kommando-String

6.2.3.2 rvs® Monitor aufwecken

Prototyp `rvsWakeMonitor`:

```
PROCDEF int PROCKEYW rvsWakeMonitor( void);
```

Beschreibung der Parameter

keine Parameter vorhanden

6.2.3.3 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_RC_WAKE_FAILED**, wenn das Aufweck-Kommando fehlschlägt

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.4 Funktionen zur Verwaltung von residenten Empfangseinträgen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von residenten Empfangseinträgen erforderlich sind.

6.2.4.1 Typ-Definitionen und Makros

```
  typedef struct{
  char  uid_local[RVSCAL_L_USID];
  char  vdsn[RVSCAL_L_VDSN];
  char  uid_sender[RVSCAL_L_USID];
  char  sid_sender[RVSCAL_L_STATID];
  char  dsn_local[RVSCAL_L_DSN];
  char  s_replace[RVSCAL_L_C1];
  char  s_disp[RVSCAL_L_C1];
  SINT  flg_stamp;
  char  s_printdef[RVSCAL_L_C1];
  char  dsn_batchjob[RVSCAL_L_DSN];
  char  uid_creator[RVSCAL_L_USID];
  char  accnt_rcv[RVSCAL_L_ACCT];
  char  comment[RVSCAL_L_RECMNT];
  char  dt_lastused[RVSCAL_L_DT];
 } INFO_RE ;


#define RE_UPDATE    1
#define RE_DELETE    2
#define RE_CREATE    3
```

6.2.4.2 Nächste Kommandonummer des residenten Empfangseintrages aus der Datenbank holen

Prototyp `rvsGetNextRE`:

```
PROCDEF int PROCKEYW rvsGetNextRE( const long
cn_pre, long *lpcn_re);
```

Beschreibung der Parameter

cn_pre	(const long, input)
	Kommandonummer des vorherigen residenten Empfangseintrages
lpcn_re	(long *, output)
	Kommandonummer des nächsten in der **RE**-Tabelle gefundenen residenten Empfangseintrages

6.2.4.3 Residenten Empfangseintrag aus der Datenbank holen

Prototyp `rvsGetRE`:

```
PROCDEF int PROCKEYW rvsGetRE( const long cn_re,
INFO_RE *reinfo);
```

Beschreibung der Parameter

cn_re	(const long, input)
	Kommandonummer des residenten Empfangseintrages
reinfo	(**INFO_RE** *, output)
	Struktur mit Informationen über den residenten Empfangseintrag

6.2.4.4 Residente Empfangseinträge konfigurieren

Prototyp `rvsResidentResceiveEntry`:

```
 PROCDEF int PROCKEYW rvsResidentReceiveEntry(
const int icmd, INFO_RE *reinfo);
```

Beschreibung der Parameter

icmd (const int, input)

Kommando zur Angabe, was getan werden soll

RE_UPDATE – überarbeitet den residenten Empfangseintrag
RE_DELETE – löscht den residenten Empfangseintrag
RE_CREATE – erstellet einen residenten Empfangseintrag

reinfo (INFO_RE *, input)

Struktur mit Informationen über den residenten Empfangseintrag

6.2.4.5 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden residenten Empfangseintrag

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_DSN**, wenn ein ungültiger **DSNNEW** angegeben wurde

=**RVSCAL_INVALID_JOB**, wenn ein ungültiger **JOB** angegeben wurde

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INVALID_SID**, wenn **SID_SENDER** nicht bekannt ist

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, residente Empfangseinträge zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_RE_NOT_FOUND**, wenn ein residenter Empfangseintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_RE**, wenn bei **icmd**=**RE_CREATE** ein doppelter residenter Empfangseintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.5 Funktionen zur Verwaltung von Einträgen für Jobstart nach Sendeversuch

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Einträgen für Jobstart nach Sendeversuch erforderlich sind.

6.2.5.1 Typ-Definitionen und Makros

```
  typedef struct{
  char  vdsn[RVSCAL_L_VDSN];
  char  uid_sender[RVSCAL_L_USID];
  char  sid_receiver[RVSCAL_L_STATID];
  long  cnt_sendatt;
  char  dsn_batchjob[RVSCAL_L_DSN];
  char  uid_creator[RVSCAL_L_USID];
  char  comment[RVSCAL_L_RECMNT];
  char  dt_lastused[RVSCAL_L_DT];
 } INFO_JS ;
#define JS_UPDATE    1
#define JS_DELETE    2
#define JS_CREATE    3
```

6.2.5.2 Nächste Kommandonummer des Eintrages für Jobstart aus der Datenbank holen

Prototyp `rvsGetNextJS`:

```
PROCDEF int PROCKEYW rvsGetNextJS( const long
cn_pre, long *lpcn_js);
```

Beschreibung der Parameter

cn_pre	(const long, input)
	Kommandonummer des vorherigen Eintrages
lpcn_js	(long *, output)
	Kommandonummer des in der **JS**-Tabelle gefundenen Eintrages für Jobstart nach Sendeversuch

6.2.5.3 Jobstarteintrag aus der Datenbank holen

Prototyp `rvsGetJS`:

```
PROCDEF int PROCKEYW rvsGetJS( const long cn_js,
INFO_JS *jsinfo);
```

Beschreibung der Parameter

cn_js	(const long, input)
	Kommandonummer des Jobstarteintrages
jsinfo	(**INFO_JS** *, output)
	Struktur mit Informationen über den Jobstarteintrag

6.2.5.4 Eintrag für Jobstart nach Sendeversuch konfigurieren

Prototyp `rvsJobStartEntry`:

```
PROCDEF int PROCKEYW rvsJobStartEntry( const int
icmd, INFO_JS *jsinfo);
```

Beschreibung der Parameter

icmd	(const int, input) Kommando zur Angabe, was getan werden soll **JS_UPDATE** – überarbeitet einen Jobstarteintrag **JS_DELETE** – löscht einen Jobstarteintrag **JS_CREATE** - erstellt einen Jobstarteintrag
jsinfo	(**INFO_JS** *, input) Struktur mit Informationen über den Jobstarteintrag

6.2.5.5 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden Jobstarteintrag gibt

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_DSN**, wenn ein ungültiger **DSNNEW** angegeben wurde

=**RVSCAL_INVALID_JOB**, wenn ein ungültiger **JOB** angegeben wurde

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INVALID_SID**, wenn **SID_RECEIVER** nicht bekannt ist

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, Einträge für Jobstart nach Sendeversuch zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_JS_NOT_FOUND**, wenn der Jobstarteintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_JS**, wenn bei **icmd** = **JS_CREATE** ein doppelter Jobstarteintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.6 Funktionen zur Verwaltung von Benutzereinträgen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Benutzereinträgen erforderlich sind.

6.2.6.1 Typ-Definitionen und Makros

```
typedef struct{
  char  uid[RVSCAL_L_USID];
  char  s_priv[RVSCAL_L_C1];
  char  s_prof[RVSCAL_L_C1];
  char  s_lang[RVSCAL_L_LANG];
  char  s_name[RVSCAL_L_USERNAME];
 } INFO_USER ;


#define USER_UPDATE    1
#define USER_DELETE    2
#define USER_CREATE    3
```

6.2.6.2 Nächsten Benutzer aus der Datenbank holen

Prototyp `rvsGetNextUser`:

```
PROCDEF int PROCKEYW rvsGetNextUser( char
*userpre, char *user);
```

Beschreibung der Parameter

userpre	(char *, input)
	vorheriger Benutzername
user	(char *, output)
	Nächster Benutzername in der BT-Tabelle

6.2.6.3 Benutzereintrag aus der Datenbank holen

Prototyp `rvsGetUser`:

```
PROCDEF int PROCKEYW rvsGetUser( char *user, IN-
FO_USER *usinfo);
```

Beschreibung der Parameter

user	(char *, input)
	Benutzername
usinfo	(**INFO_USER** *, output)
	Struktur mit Informationen über den Benutzereintrag

6.2.6.4 Benutzereintrag konfigurieren

Prototyp `rvsUser`:

```
PROCDEF int PROCKEYW rvsUser( int icmd, INFO_USER
*usinfo);
```

Beschreibung der Parameter

icmd	(const int, input)
	Kommando zur Angabe, was getan werden soll
	USER_UPDATE – überarbeitet einen Benutzer **USER_DELETE** – löscht einen Benutzer **USER_CREATE** – erstellt einen Benutzer
usinfo	(**INFO_USER** *, output)
	Struktur mit Informationen über den Benutzereintrag

6.2.6.5 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden Benutzereintrag gibt

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_OWN_PRIV**, wenn der Benutzer seine Privilegien senken will

=**RVSCAL_INVALID_UID**, wenn der Parameter **UID** leer ist

=**RVSCAL_INVALID_USER**, wenn der Benutzer nicht vorhanden ist oder sich selbst zu löschen versucht

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, Benutzereinträge zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_USER_NOT_FOUND**, wenn der Benutzereintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_USER**, wenn bei **icmd** = **USER_CREATE** ein doppelter Benutzereintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.7 rvs® Datenbank Funktionen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Datenbankfunktionen erforderlich sind.

6.2.7.1 Typ-Definitionen und Makros

```
#define RVSCAL_PIPE_NAME "\\\\.\\pipe\\rvsdb"
#define RVSCAL_OLEVENT_NAME "rvsdb_olevent"
#define RVSCAL_PIPE_TIMEOUT  90000
#define RVSCAL_L_OLEVENT    14
#define RVSCAL_L_PIPENAME   15
#define DEL_DB          0x01
#define DEL_LOG         0x02
#define DEL_TMP         0x04
#define DEL_REMDB        0x08
#define DUMP_RES         0x01
#define DUMP_USER        0x02
#define DUMP_JS         0x04
#define DUMP_STATION       0x08
#define DUMP_RU_ALL      DUMP_RES | DUMP_USER |
DUMP_JS | DUMP_STATION
```

6.2.7.2 Datenbank speichern

Prototyp `rvsDumpDB`:

```
PROCDEF int PROCKEYW rvsDumpDB( char
*environment, char *dsn);
```

Beschreibung der Parameter

environment	(char *, input)
	Name der Umgebungsdatei
	Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung
dsn	(char *, input)
	Dateiname der zu speichernden Datei

6.2.7.3 Datenbank wiederherstellen

Prototyp `rvsWriteDB`:

```
PROCDEF int PROCKEYW rvsWriteDB( char
*environment, char *dsn);
```

Beschreibung der Parameter

environment	(char *, input)
	Name der Umgebungsdatei
	Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung
dsn	(char *, input)
	Dateiname der zu gespeicherten Datei

6.2.7.4 Datenbank initialisieren

Prototyp `rvsInitDB`:

```
PROCDEF int PROCKEYW rvsInitDB( char
*environment, char *sid_loc);
```

Beschreibung der Parameter

environment	(char *, input)
	Name der Umgebungsdatei
	Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung
sid_loc	(char *, input)
	lokale Stations-ID

6.2.7.5 Datenbank löschen

Prototyp `rvsDeleteDB`:

```
PROCDEF int PROCKEYW rvsDeleteDB( char
*environment, const int delattrib);
```

Beschreibung der Parameter

environment (char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung

delattrib (const int, input)

Das Attribut gibt an, welche Datei gelöscht werden soll. Mögliche Werte sind:

DEL_DB (Standard) – alle Datenbankdateien löschen,

DEL_LOG – Logdateien entfernen,

DEL_TMP – alle Dateien aus dem temporären rvs® Verzeichnis entfernen

6.2.7.6 Benutzer-, Empfangs-, Jobstart- und Stationseinträge speichern

Prototyp `rvsDumpRU`:

```
PROCDEF int PROCKEYW rvsDumpRU( char
*environment, char *dsn, const int dumpattrib);
```

Beschreibung der Parameter

environment (char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nulltermi-

nierter String ist, sucht rvs® nach der Standardumgebung

dsn (char *, input)

Dateiname der zu speichernden Datei. Die Dateiinhalte erhalten das für die rvs® Kommandozeilen-Schnittstelle einsetzbare Eingabeformat.

dumpattrib (const int, input)

Das Attribut gibt an, welche Datei wiederhergestellt werden soll. Mögliche Werte sind:

DUMP_USER (Standard) – alle Benutzereinträge speichern,

DUMP_RES – alle residenten Empfangseinträge speichern
DUMP_JS – alle für Jobstart nach Sendeversuch speichern
DUMP_STATION – alle Stationseinträge speichern

6.2.7.7 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_DSN_NOT_EXIST**, wenn die angegebene Datei nicht vorhanden ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

6.2.7.8 Versionsnummer der rvs® Datenbank lesen

Prototyp `rvsGetVersion`:

```
PROCDEF int PROCKEYW rvsGetDBVersion( char
*pszDBVersion);
```

Beschreibung der Parameter

pszDBVersion (char *, output)

Version der rvs® Datenbank

6.2.7.9 Rückgabewerte

FUNCTIONVALUE (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_ENVIRONMENT_NOT_EXIST**, wenn die Umgebungsdatei nicht vorhanden ist

=**RVSCAL_ERROR_GETVERSION**, Version kann nicht bestimmt werden

6.2.8 Andere Funktionen

Dieses Kapitel beschreibt, wie Sie SID aus der ODETTE-ID erzeugen und umgekehrt und wie Sie den Listenstatus der rvs® Kommandos erhalten.

6.2.8.1 SID aus der ODETTE ID erzeugen oder umgekehrt

Prototyp `rvsgetsid`:

```
char *rvsgetsid(char *s_odette_id);
```

Beschreibung der Parameter

FUNCTIONVALUE	(char *)
	=**NULL**, wenn kein **SID** gefunden wurde oder bei einem **DB** Fehler.
	=Zeiger zum String, der rvs® **SID** enthält
s_odette_id	(char *, input)
	Zeiger zum String, der die ODETTE ID (max. 26 Bytes) **SID** enthält

Prototyp `rvsgetodid`:

```
char *rvsgetodid(char *s_sid);
```

Beschreibung der Parameter

FUNCTIONVALUE	(char *)
	=**NULL**, wenn keine ODETTE ID gefunden wurde oder bei einem **DB** Fehler
	=Zeiger zu einem maximal 26 Bytes langen String
s_sid	(char *, input)
	Zeiger zum String, der rvs® **SID** enthält

6.2.8.2 Auflisten der Status der rvs® Kommandos

Prototyp `rvslistcmd`:

```
typedef struct {
  char status;
  short errorcode;
  } RVSCMD;
int rvslistcmd(long l_cmdid, RVSCMD *p_info);
```

Beschreibung der Parameter

FUNCTIONVALUE	(int) =**RVSCAL_OK**, wenn `rvslistcmd` erfolgreich ist
l_cmdid	(long, input) Die **CMDID** (Kommandonummer) des Kommandos, welches bearbeitet wurde (Rückgabewert von `rvscal`).
p_info	(struct **RVSCMD** *, output) Zeiger auf eine Struktur, welche Informationen über das Kommando **CMDID** enthält.
RVSCMD.status	(char, output) Zeichen, das den Status des Kommandos enthält: **a/d/e/h/p/q/s/f/...**
RVSCMD.errorcode	(short, output) Short Integer, welche einen Fehler-Code enthält, wenn dieser von **0** abweicht**.**