T · · Systems · · ·

rvs® portable

Version 5.07

Reference Manual

rvs® portable

Version 5.07

Reference Manual

© 2012 by T-Systems International GmbH

Holzhauser Straße 4 – 8

13509 Berlin

Contents

Change History

The following changes of Reference Manual were made in the previous releases (including the current release):

Version 5.07:

- New parameter in chapter 8.1 “rvs® Parameters′ Overview”: `RVSDIAEXTENDEDMODE`
- New value "D" for `VFTYP` parameter
- New `rvssce` parameters: **-f** and **-i**

Version 5.06:

- New parameter in chapter 8.1 “rvs® Parameters′ Overview”: `CHECKMAXPSESSIONS`
- Now it is possible to send/receive Comsecure compressed and encrypted files > 2GB
- On Windows systems rvsjs can be started with option `–t` for supervising the job directory periodically
  On UNIX systems configuration of rvsjs can be done also on the command line
- Little changes

Version 5.05:

- New parameters in chapter 8.1 “rvs® Parameters′ Overview”: `AUTODECRYP`, `CALLINGNUMCHECK`, `CNTIE`, `DTCOPY`, `EFIDGAPTIMEOUT`, `HEAVYDUTY`, `IECLEANTIME`, `STATCHECKINT`.

Version 5.04:

- Little changes

Version 5.03:

- Chapter 7.2 “CISCO Configuration” changed

Version 5.00:

- New parameters `DIALCOUNT` and `DIALRTIME` in chapter 8.1, important for the feature “Alternative Networks”
- New parameters `REDOMAXSIZE` and `NUMREDOLOGS` in chapter 8.1 important for the feature “Backup/Recovery”.
- New parameter `TRACEOFF` for possibility to write the line traces into memory, see chapter 8.1.

- New parameters `MWSTART`, `MWSTOP` and `MWTIMEOUT` important for the reliability of rvs® Client/Server (chapter 8.4 in rvs® Client/Server User Manual and 8.1 chapter in this manual)
- New parameters `DBLOGMAXENTRIES` and `DBLOGMINENTRIES` see chapter 8.1
- New VFTYP parameter values X and U for SEND command in rvsbat (see chapter 11.4)

Version 4.05

- Chapter 7 "XOT Router Configuration"
- Chapter 2.14 "The robustness of rvs® concerning disk space" with new parameters `DISKSPACEERR`, `DISKSPACEWARN`, `DISKSPACEDELAY` and `DISKSPACEREJECT`.

Version 4.00:

- Command substitution patterns ?CNIE? and ?CNIZ? in RE (chapter 11.5 "Command RESENTR");
- Chapter 2.13 "Memory line traces "

Version 3.05:

- Chapter 2.5 "rvs® Service Provider",
- Chapter 7.10 "rvscheckdb" and
- Chapter 2.11 "rvs® Data Center".

I. Introduction

The product rvs® is available on many different platforms. The products rvsX, rvsNT, rvsXP and rvs400 form together the product group rvs® portable.

For the normal usage of rvs® you should look into the user manuals. They are provided in different platform-specific versions.

The things in common of all rvs® variants from the product group rvs® portable are described in this manual. It contains more technical details about the basic rvs® portable functionality, too.

This chapter presents a short description of the rvs® system, as well as an explanation of the notations, which are used throughout this manual.

1. rvs® and its interfaces

rvs® provides an efficient and reliable transport service for files of any format or content.

rvs® can be integrated into applications for the automation of data exchange. Typical areas of rvs® application are EDI (Electronic Document Interchange), CAD (Computer Aided Design), financial transaction systems, safe transmission of master data and data of media companies.

For different degrees of automation are available suitable rvs® interfaces:

Dialog Interface `rvsdia`	Is an interactive tool for the creation of the rvs® individual transfer tasks; inquiry functions inform you about the task status.
Command Line Interface `rvsbat`	Reads commands from a file. The input file is a simple text file, which can be prepared with any editor or be produced as output file of an application program.
C-CAL-Interface `rvscal`	Enables application programs to generate rvs® command entries by calling functions of the programming language "C".
J-CAL-Interface	This interface was developed in connection with the rvs® Client/Server, a networkable extension of rvs® portable. It enables application programs to generate rvs® command entries by calling methods of the programming language Java.
XML-Interface	makes possible export and import of rvs® related data in the XML format.

1.1. Representation means

This chapter contains the description of the indications which are used in this manual and the explanation of the expressions which are marked.

Indications

`courier`	commands, menu commands, file names, path names, programs, examples, scripts, qualifiers, data sets, fields, options, modes, window names, dialog boxes and statuses
BOLD and IN CAPITAL LETTERS	parameters, environment variables, variables
"quotation mark"	links to other manuals, sections and chapters, literature
bold	important, names of operating systems, proper names, buttons, function keys

Expressions

rvsX is the synonym of rvs® for **UNIX** systems.

rvsNT is the synonym of rvs® for **Windows NT** systems.

rvsXP is synonym of rvs® for **Windows XP / 2000 / Vista / 7 / WS 2003 / WS 2008** Systemen

rvs400 is the synonym of rvs® for **OS/400** systems.

Directories

As user directories are found on different locations for the different operating systems we use the variable **$RVSPATH** in this manual. Default values are:

- `/home/rvs/` for **AIX**, **Solaris**, **IRIX**, **Linux** and **SCO**
- `/users/rvs/` for **HP-UX**
- `/defpath/rvs/` for **SINIX**
- `c:\rvs` for **Windows**

Substitute the variable with your correct path.

Generally, the file names on **OS/400** systems are always written in capital letters.

II. Technical Overview

This part gives an overview of the functional elements of rvs® and the protocol layers in the rvs® communication, as well as the basic concepts of the LU 6.2, the X.25 native communication, and TCP/IP in regard to rvs®.

2. Functional Elements

Elements of portable rvs®

rvs® consists of the following main elements:

- Monitor
- MasterTransmitter (`rvsxmt`)
- Communication Modules
- LogWriter
- ActivePanel
- Operator Console (`rvscns`)
- Dialog Interface (`rvsdia`)
- Batch Interface (`rvsbat`)
- C-Cal Interface (`rvscal`), J-Cal Interface, XML Interface
- Database

Information Flow

In a running rvs® environment, the central focal point is the rvs® database which keeps all necessary static and dynamic information. Static information is for example the table of rvs® parameters and the table of stations that can be accessed from the local node. Dynamic information is information about current processes like send orders.

The work flow, i.e. the exchange of control information between all related tasks is organized in a loosely coupled mode. The tasks do not directly communicate with each other (exception: the console task, which in some implementations of rvs® uses pipe communications with the Monitor), they rather use the rvs® database as communication medium.

A close coupling in the sense of shared memory and subtasking - as it is the case in rvsMVS - is not possible because of portability and maintainability reasons. The aim was to use as much identical code as possible across all supported platforms. Only few systems support subtasking and shared memory, and if so, the code cannot be kept portable. However, multiprocessing and external data are common to all system, except for PCs under MS DOS or PC-DOS.

In order to efficiently master the continuously growing flow of data and to uncrease the performance level of rvsX and rvsNT, from the version 2.05 and above it is possible to bind to an Oracle database. The rvs® internal C-ISAM database is replaced with the external high performance Oracle database. For rvsNT and rvsXP from the version 2.11 there is besides Oracle also the possibility of binding to a Microsoft SQL Server.

2.1. Monitor

This chapter describes the monitor basic characteristics, the processing of a send order and the handling of incoming data.

2.1.1. Monitor: Basic Characteristics

The Monitor is the agent who dispatches all work to be done and who reacts on external events. The Monitor is the central component of rvs®. Its main duties are to

- dispatch work,
- process operator commands,
- scan the database for send orders to be processed,
- submits a MasterTransmitter process, if not already active and if something is ready to be transmitted,
- reinitialize failed transmissions,
- deliver incoming information to the final destination (user or another node)
- activate batch jobs if a matching resident receive entry is found upon reception of a data set,
- write statistics records,
- create user notifications,
- modify parameters and stationtable entries in the database upon operator request,
- recover the database.

The Monitor is controlled via an operator console task. Depending on the local system, the console task is tightly coupled to the

Monitor via a thread or pipe (e.g. OS/2), or it is loosely coupled by placing its orders into the database (OS/400 or UNIX).

The Monitor periodically scans the database for external events or processable units of work. If nothing is found to be done, the Monitor suspends itself for a user-defined period of time (rvs® parameter **SLEEP**). External events like the submission of an operator command or the creation of a new send order are signalled to the monitor via a semaphore or similar, causing its immediate `wakeup'. Whatever unit of work the Monitor has found to do, it converts it to an rvs® command of the correct type to which a unique command number is assigned. The further processing of each rvs® command is controlled by correspondent command status and priority information. Each kind of internal command has assigned a certain priority that can be modified by the rvs® operator.

The basic function of the Monitor is to deal with rvs® commands and related statuses. The most important commands or events are:

`SE`	send order entry (German: `SendeEintrag')
`SK`	send command (`SendeKommando')
`QS`	send command for receipt (`QuittungsSendung')
`QE`	received receipt (`QuittungsEingang')
`IE`	information entry (incoming information, `InformationsEingang')
`IZ`	information delivery to recipient (`InformationsZustellung')
`OK`	operator command (`OperatorKommando')
`EC`	command for `EndCommand' processing (cleanup)

The list of possible statuses is:

q	queued, awaiting processing by monitor
`f`	forwardable, awaiting processing by MasterTransmitter
a	active, processing by monitor underway
`i`	in transit, processing by transmitter underway
p	pending
e	ended
d	logically deleted from database
h	held by system or by operator
s	all traffic to destination has been suspended (`SK`, `QS`)

More or less all commands go through the following chain of statuses: `queued' or `forwardable', `active' or `in transit', (`pending'), `ended'.

2.1.2. Processing of a Send Order

A user has created a send order entry SE in the database. Upon next scan of the database it is found, interpreted and analyzed. A processable unit of work, a send command SK is created.

The SE is now placed in status `forwardable'. The Monitor will activate the MasterTransmitter process if this has not already been done before. Execution of the SK results in submission of a Sender task by MasterTransmitter.

The status of the SK is now `in transit', the corresponding SE is placed in status `pending'. The completion of the Sender task, successful or not, is notified to the Monitor via the database. Along with a readable notification message, the Sender updates the status and error fields of the SK.

The Monitor will display the notification message to the console and will initiate a restart if the error field of the SK is indicating a failure. If successful the SK will be placed in status `ended', the SE is still in status `pending'. When eventually the receipt, QE, which is the ODETTE end-to-end response for successful transfer, will be received from the remote partner, the Monitor will mark the originating SE as `ended'.

2.1.3. Handling Incoming Data

A Receiver task is initialized from the remote station if SNA LU 6.2 communications technics are used.

In other cases, i.e. X.25 native, a set of prestarted receiver tasks is waiting for incomming calls. The completion of the receive process is also indicated in the database by creating an incoming information event entry IE.

The incoming data are stored in a temporary data set. Upon detection of the IE event, the Monitor initiates the appropriate action, i.e. the delivery of the data set to the destination recipient, there might be more than one, by creating an IZ internal command. Usually, the processing of the IZ consists of copying the received data from the temporary data set and placing itself and the related IE in status `ended'. If the IE contains information, that has to be forwarded to another rvs® node, a SE is created for the required transfer instead of an IZ.

2.2. MasterTransmitter

The MasterTransmitter has been introduced for better control of parallel transmission processes. It's main functions are:

- control of total maximum number of active transmitters (outbound communication processes) in order not to overload a local system
- to queue outbound transmissions
- to submit transmitters if maximum number of transmitters is not exceeded
- control of maximum number of active transmitters per remote station. This future extension will allow to handle station specific ressource limitations like the available number of SVCs in a X.25 native connection.
- a station dependent intelligent mechanism that allows to send more than one data set, or even to receive data sets during one transmission process.
- control of prestarted receiver processes, e.g. for X.25 native support.

2.3. Communication Modules

Sender and receiver processes are consolidated into a common communication program, which acts as sender or receiver depending on call parameters. The role as sender or receiver may dynamically be changed. The communication program consits of four hierarchical layers:

- layer 1 (top): the communication module (main program),
- layer 2: the sender and receiver modules,
- layer 3: the ODETTE File Transfer Protocol (OFTP) module,
- layer 4: the linedriver module.

The communication process is activated as a sender by MasterTransmitter with the number of the respective send command as input parameter. Using this number, the communication module queries the database for the necessary information, i.e. remote station ID, data set name, LU 6.2 or X.25 and ODETTE parameters, and starts the transmission. After sending, it asks the partner to send his queued data sets.

After transmission, the sender looks for other data sets to send; that means, if several data sets are queued for sending to the same station, they all will go over only one established connection.

The communication process is activated as receiver upon an incoming LU 6.2 APPC remote program-call, or it has been prestarted, waiting for incoming X.25 or TCP/IP calls. The necessary information about the calling station is obtained from the ODETTE protocol station ID and Password exchanged in the first OFTP header records and will be counterchecked for authorization. Information about the incoming data like data set name, file size and record type, is derived from the initial header records exchanged. If a Receiver has successfuly terminated, it places an `IE` entry in the database, which the Monitor will find to initiate the delivery of the data, `IZ`, to the final destination.

The communication process can be activated by an "ACTIVATE" command. Then it acts like a sender and connects to the desired station. If there are queued data sets on the local station or the remote station, they will be transmitted. Otherwise, the connection ends.

2.4. LogWriter

The Logwriter is the central module responsible for gathering information about data transfers from the other rvs® processes. The Information interchange between the LogWriter and the other rvs® processes takes place via sockets. This functional detail requires that the TCP/IP protocol stack must be activated (for Windows, if necessary via a Loopback adapter).

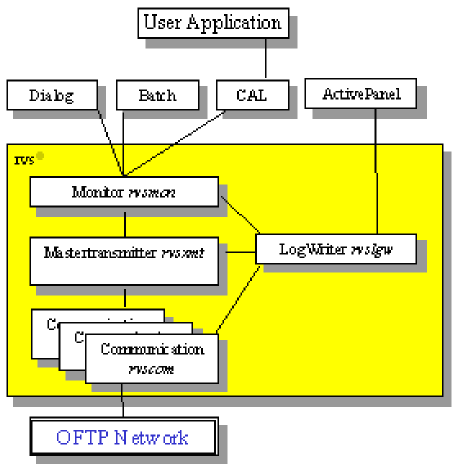
The transfer information is divided in two types: status information and event information. Status information is information about the development of the currently active data transfer, e.g. how many bytes are being transmitted at the moment, or whether a transmission error is occurring, etc.

Event information specifies e. g. when a file has been transferred or which files have been transferred. This type of information refers to events which have taken place already.

The information is displayed differently depending on the type:

- You can call and display status information about current data transfers in the ActivePanel.
- The information about events which have taken place between the rvs® start and stop is written in a temporary message file. You can call and display the event information via the Operator Console.
- Information about events which are independent from the rvs® start or stop are saved in a specially defined message file. It can be displayed by means of an ASCII editor.

The following picture exemplifies the functions of the LogWriter:



2.5. rvs® Service Provider (rvsSP)

rvs® Service Provider is an internal rvs® application providing the following services:

- Compression and decompression
- Encryption and decryption.

If necessary, files are compressed and/or encrypted prior to transmission and decompressed and/or decrypted after reception.

`rvsSP` closely interacts with the rvs® Monitor. rvs® Monitor will continue controlling send/receive jobs in rvs® and uses `rvsSP` for pre- and post-processing of files (prior to transmission or after reception).

rvs® Monitor writes all details necessary for a job to `rvsSP` into one file per job – the job file. rvs® Monitor saves the job file in the `SPINDIR` folder. Then `rvsSP` processes all jobs contained in `SPINDIR`. Intermediate files created by `rvsSP` are saved in a separate `SPFILESDIR` folder. `rvsSP` saves processed jobs in the `SPOUTDIR` folder. rvs® Monitor is informed via an IP socket connection and then processes all jobs contained in `SPOUTDIR`.

Note: Set paths for the `SPINDIR`, `SPFILESDIR` and `SPOUTDIR` variables in the `$RVSPATH/rvsenv.dat` file. See User Manual, chapter 3.11 for an explanation of `SPINDIR`, `SPFILESDIR` and `SPOUTDIR`.

2.6. ActivePanel

The ActivePanel is a display programme, which is used to display information about currently active data transfers.

Start the ActivePanel:

- using the `rvsap` programme for **UNIX**
- using the main window `rvs Administrator` menu `View → Active lines` for **Windows**

2.7. Operator Console (rvscns)

The operator interface is used to control operations of rvs®. It provides - via operator commands to the Monitor - an interface for manipulations of the database. Some of the dynamic information, for example send commands and send orders, can be displayed and modified. Other commands allow the display and modification of rvs® parameters and the stationtable. The modifications for the stationtable are read from an input data set.

2.8. Dialog Interface (rvsdia)

The rvs® Dialog Interface is a `look similar' approach to the well known menu system of rvsMVS. It allows entry, modification and deletion of send orders and resident receive entries and the display of statuses. The Dialog Interface communicates directly with the rvs® Database.

The Dialog Interface consits of a finite state machine, which is common to all portable rvs® implementations on the various platforms.It has a set of display, entry and help panels or menus, and the database interface. The panels are portable across all systems supporting ANSI Terminals. Only on AS/400 the native panel systems has been used.

2.9. Batch (rvsbat) and Call (rvscal) Interface

The Batch Interface allows to enter orders as a single command line on the respective systems command language level. Such a command can easily be included in command list procedures. It can be used either interactively, or in a command list data set which might be executed in batch mode.

The Call Interface can be linked directly into a user application program. It allows to place orders into the rvs® Database out of a user application program. A sample C Program and some batch samples are included in the rvs® distribution.

2.10. Database

The rvs® Database is a relational database supporting the SQL query language. All rvs® programs use "embedded SQL" calls. The rvs® Database is organized in a set of tables which are listed below:

AC	table for X.28/PAD or ASCII parameters
BB	table for user notifications (German: "BenutzerBenachrichtigung")
BT	table of locally registered rvs® users ("BenutzerTabelle")
CT	table of valid console-ids
DB	table for actual rvs® and database version
EC	table of "EndCommand" type internal commands
ET	table of valid receivers ("EmpfaengerTabelle")
FK	table of command errors
FS	table of station errors
IE	table for information about received transmission ("InformationsEingang")
IZ	table for information about deliveries ("InformationsZustellung")
JS	table for job starts after send attempts
KT	table of commands being processed ("KommandoTabelle")
LC	table containing the last unique command number ("LastCommand")
LD	LastDate (last used ODETTE-time)
LM	table for Log Messages
LU	table for LU 6.2 parameters
LT	table of Data Center Log Messages
LX	table for linedriver parameters X-tensions.
NK	table for neighboring nodes ("NachbarKnoten")
OK	table for Operator Commands
OP	table for ODETTE Parameters
PT	table for rvs® parameters ("ParameterTabelle")

QE	table for received receipts ("QuittungsEingang")
QS	table for receipts to be sent ("QuittungsSendeeintrag")
RE	table for Resident receive Entries
RI	table of rvs® information (rvs® Data Center)
RT	table for routing infomation ("RoutingTabelle")
SE	table for send order entries ("SendeEintrag")
SK	table for send commands ("SendeKommando")
SL	table for Serialization Lists
SS	table for Send Statistics
ST	table for stations ("StationsTabelle")
TC	table for TCP/IP parameters
VD	table for info about data set to send ("VersandDatei")
VM	table for send notifications ("VersandMeldung")
XP	table for X.25 native parameters

The primary key of a table usually is the station ID "SID". The tables are organized in rows and colums. The rows contain one command or station description or whatever, the columns contain the various details of informatin like status, name of data set, date and time of entry etc..

The physical database depends on the local system. rvs® currently supports Oracle, MS SQL and ISAM databases. The physical differences are hidden under a embedded-SQL interface which is common throughout all portable rvs® versions. This is of major importance for the portability and maintainability of portable rvs® since the logic design can be kept completely independent of the physical file or database system which can vary considerably from system to system.

In order to efficiently master the continuously growing flow of data and to increase the performance level of rvs®, rvs® 2.06 and above on Windows NT, XP, 2000, Vista, 7 WS 2003, WS 2008, AIX, Linux and Sinix Systems gives the possibility of binding to an ORACLE database and rvs 2.11 and above on Windows Systems gives the possibility of binding to Microsoft SQL database. The rvs® internal C-ISAM database is replaced with the external high performance Oracle or MS SQL database.

The database entries can be printed with the rvs® Database dump command `rvsddb`.

2.11. rvs® Data Center

The present chapter describes the technical basis of rvs® Data Center; its operation is explained in the rvsX 3.05 User Manual.

2.11.1. Introduction

rvs® Data Center offers significantly higher fail safety and transmission capacity than rvs®.

An rvs® Data Center comprises several rvs® servers aiming at ensuring high system availability.

Jobs to be processed are evenly distributed among all rvs® servers within the rvs® Data Center (load balancing).

Transmission capacity can be increased or decreased by adding or removing rvs® servers at rvs® Data Center run time (scalability).

To ensure trouble-free rvs® Data Center operation another server can assume the tasks of a failed server.

With regard to communication partners and operation (e.g. file transfer), rvs® Data Center behaves like a single rvs®.

2.11.2. System requirements

For rvs® version 5.06.00, rvs® Data Center is available for the following platforms:

- Window XP / 2000 / 2003 / Win7 / Vista / 2008
- AIX 5.3 / 6.1
- Linux i686
- HPUX ia64 / risc
- Sun SPARC Solaris.

The following Versions of Oracle or MS SQL Server are used as rvs® database:

• Windows:	MS SQL-Server (all versions), Oracle 10, Oracle 11*)
• AIX 5.3 / 6.1:	Oracle 10, Oracle 11
• Linux i686:	Oracle 9, Oracle 10, Oracle 11*)
• HPUX ia64:	Oracle 9, Oracle 10
• HPUX risc:	Oracle 9, Oracle 10*^{2}), Oracle 11*^{2})

- Sun SPARC Solaris: Oracle 10

*) use Oracle 10 Client

*²) use Oracle 9 Client

To ensure access to the Oracle database, Oracle client software must be installed on each rvs® server (node).

The NFS (Network File System) protocol version 3 is required to access the shared directories of the rvs® Data Center over the network.

2.11.3. rvs® Data Center architecture

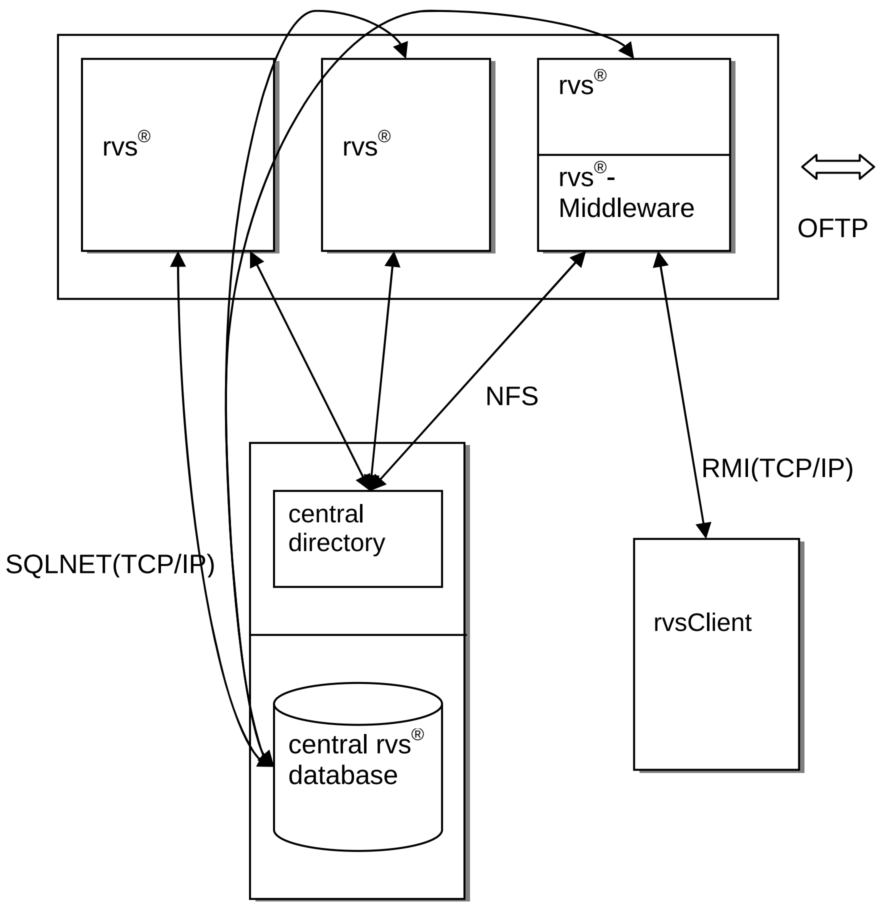
rvs® Data Center comprises the following components:

- several rvs® nodes (rvs® servers with rvs® version 3.05 and later as well as Oracle client software installed).
- a central Oracle database (version 8.1.7) the rvs® servers can reach via `SQLNET` using TCP/IP.
- a central directory holding the most important directories used by rvs® Data Center. This central directory must be accessible by every rvs® server via NFS (Network File System). We recommend that you set up this directory on a separate computer. This directory could be located on the same computer holding the central database but, in order not to impair the entire system’s fail safety, not on one of the rvs® nodes. The central directory must contain the following subdirectories:
  - `temp`: for temporary file storage while transmission and reception is active.
  - `usrdat`: for delivery of received files.
  - `init`: for configuration files and license.
  - `keydir`: for the file encryption and decryption keys.
  - `spindir`: for the internal service provider job files (compression and encryption) that are to be processed.
  - `spoutdir`: for the internal service provider job processing log files. You can use the log files for error analysis.
  - `spfilesdir`: for the internal service provider job files used during job processing.
  - All files to be sent must also be located in the central directory.
- rvs® Client/Server (which in turn comprises rvs® Middleware and rvs® Client). rvs® Client/Server is used to configure and maintain the rvs® Data Center. rvs® Middleware must run on any node while rvs® Client can be executed on a computer outside the rvs® Data Center. Communication between Client

and Middleware is implemented by an RMI connection (Java). In case a firewall is installed between rvs® Client and rvs® Middleware, seven freely configurable ports must be opened.

rvs® Client/Server versions:
rvs® version 5.06 is released with rvs® Client/Server version 5.05.00.
The combination of rvs® 5.06 and rvs® Client/Server (Client API) 5.05.00 is compatible downward up to rvs version 3.04 in combination with rvs® Client/Server 2.02.

The following illustration shows the rvs® Data Center architecture:



Connecting networks (ISDN, X.25 or TCP/IP) for file transmission or reception must be set up on all rvs® nodes. The above illustration shows this component with OFTP.

2.11.4. rvs® Data Center user interface

The rvs® Data Center user has access to the following interfaces:

- rvs® Batch interface (`rvsbat`)
- scripts
- rvs® Client/Server.

`rvsbat` is primarily used for automatically sending files and for creating resident receive entries (REs) and job starts after send attempts (JSs). `rvsbat` can be run on any rvs® node. The procedure is identical as with rvs® standalone; files to be sent must be located in the central directory. Furthermore, `rvsbat` must be able to access the central database. For more information on `rvsbat`, in particular on the `SEND`, `RESENTR` und `SENDJOB` commands please refer to chapters 11.4, 11.5 and 11.6.

Note: The term rvs® standalone is used in contrast to rvs® Data Center and identifies a single rvs® (e.g. rvsXP or rvsX).

You can use scripts to start post-processing (RE or JS) or start or stop rvs® Data Center.

The graphical rvs® Client user interface allows for the following actions:

- rvs® Data Center configuration at runtime,
- job, station and user administration,
- display of log messages and statistics information with filter options
- snapshot display of rvs® Data Center configuration.

2.11.5. New rvs® Data Center system components

The following new components distinguish rvs® Data Center from rvs® standalone.

Fail safety

All rvs® components (`rvscom`, `rvsmon`, `rvsxmt`, ...) run on all rvs® nodes. Every monitor (`rvsmon`) monitors all components depending on him on the own node and the monitors on other rvs® nodes. A database table in the central database logs the activities of all nodes. Failure of a component on an rvs® node causes this component to be restarted on the respective node and the aborted job to be terminated. When a monitor fails (which a remote monitor can detect by checking the activities in the database table), rvs®

will be restarted on this very node. The restart will be initiated by the monitor that detected the activity failure in the database.

There is no monitoring of the central database and of the central directories. rvs® Data Center cannot continue operation when one or both items are unavailable. A configurable interval allows you to define for how long every monitor attempts to reach the failed component. The monitor executes an error script allowing the error to be made public if the interval is exceeded.

Note: Time needs to be synchronized on all rvs® Data Center nodes in order to ensure correct functioning of the monitors and tracing of log messages. Time synchronization is a requirement to the system where the rvs® Data Center is installed.

The following new global parameters are used to configure the monitoring mechanism: `MONTIMEOUT`, `COMTIMEOUT`, `CNTMA`, `CNTGC` and `RECERREX`.

There are also the following new parameters that must be configured in the `rvsenv.dat` rvs® environment file: `RVSNODENAME`, `SPERRTO`, `LOGINDB`, `LOGFORMAT`, `DBDL` and `DBTO`.

Please refer to chapter 12 of the rvsX 3.05 User Manual on how to configure the new rvs® Data Center parameters.

Load balancing

Each rvs® node can assume all tasks, which makes it possible to evenly distribute the load among all nodes. A monitor on a node with low load will start processing a new job earlier than a monitor with a high load because it will access the database earlier to check for new jobs to be processed (first come, first serve).

Note: The hardware upstream of the rvs® Data Center (e.g. Brick for ISDN or switch for TCP/IP) is also responsible for load balancing of incoming connections but does not form part of the rvs® Data Center.

Scalability

You can use rvs® Data Center to add new rvs® nodes or to remove old ones. This quantitative node scalability allows an rvs® Data Center to process a significantly greater amount of data and jobs than an rvs® standalone. Adding or removing nodes at runtime allows the processing capacity to be dynamically adapted without having to stop rvs® Data Center. The number of nodes that can be feasibly used depends on the environment in use (computer, network bandwidth, file system, database, etc.).

Log messages

Apart from the conventional option of writing log messages to the log file (`rlog.log`, `rlco.log`), rvs® Data Center allows the log messages of all rvs® nodes to be directly written to the database. The sequence of log messages is determined by a process type and a process ID.

Example (log message):

```
O:  2004/12/15  15:23:28{node1}[C49944][S0000000856]<CONNECT_IND        >
Recipient:  Connection  with  Station  'FRC10'  with  Credit=99,  Odette
Buffer=2048, OFTP Compression Established.
```

In this example the connection with station `FRC10` was established by rvs® node `node1` on 12/15/2004 at 15:23:28. In `[C49944]`, `C` is the process type, and `49944` the process ID. `C` stands for the rvs® communication process (`rvscom`).

Note: For a detailed syntax description of log messages please refer to the rvs® “Messages and Return Codes” manual.

The log messages can be evaluated by rvs® Client/Server, which can read the log messages from the database using filters (`Admin -> Log Messages` window) or by external applications that read the required data directly from the appropriate database tables. A database script (`export_lt.sh`) allows log messages to be exported from the database to a file.

Parameter changes at runtime

`rvsbat` or rvs® Client/Server allow the following parameters to be edited during runtime using the `setparm` command: `ODTRACLVL`, `LITRACELVL`, `SIDTRACE`, `STATISTICS`, `CMDDELETE`, `DTCONN1-20`, `TCPIPRCV`, `MAXX25RCV`, `OCREVAL` and `OEXBUF`.

To edit any other parameters the rvs® Data Center must be stopped and restarted because it is vital that these are identical on all nodes.

2.12. Data Set Names

In this manual, we use the terms `file` and `data set` as synonyms.

The syntax for valid data set names is operating system specific. "Equivalent" names may be specified as

`/myid/invoice.dat`	under UNIX
`c:\myid\invoice.dat`	under OS/2 on a PC and under Windows
`MYID/DATA(INVOICE)`	under OS/400
`MYID.INVOICE.DATA`	under MVS on an IBM host
`. . .`	

These names are case sensitive on some systems (e.g. UNIX) and may or may not be case sensitive on others. Security systems (such as RACF under MVS) may impose additional constraints on what is considered a legal data set name on a particular local system.

These differences may cause problems or inconveniences as described below when a file is sent to a different operation system.

Virtual Data Set Name

For transfer and delivery, a file is identified by its Virtual Data Set Name (**VDSN**)[1]. You can specify this **VDSN** in the **NEW DSNAME** field when you interactively create a send request (see user manual) or in the **DSNNEW** parameter of the `SEND /CREATE` command (see section 11.4 "Command `SEND`"). When you do not specify a **VDSN**, rvs® uses the name of the sending file to generate one.

When a transmitted file is being delivered to its recipient, rvs® uses the **VDSN** as one criterion to look for matching resident receive entries. If none is found or the (best) matching entry does not define a data set name, **VDSN** is also used to generate a local data set name under which the file will be stored:

rvsMVS uses **VDSN** as-is; so, when sending to an MVS host,

[1] VDSNs are used in the ODETTE file transfer protocol to pass the file name to the next node. Maximum length
(26 characters) and character set used when generating a default VDSN are due to the ODETTE protocol.

make sure to specify a **VDSN** that adheres to MVS naming conventions and that starts with a high level qualifier that RACF is happy with. Do not use quotes (**"**) or apostrophes (**'**) to delimit your data set name.

portable rvs	uses **VDSN** to generate a name for the individual data set; the name of the path or library where this file will be stored is taken from the local rvs® environment (talk to your local rvs® administrator or see "User Manual" for more information on rvs® configuration).

VDSN is also one criterion to specify a job that should execute after a send attempt (see chapter 11.6 "Command `SENDJOB`" for more informations).

Time Stamping

When time stamping is requested, rvs® generates unique data set names when it delivers the file by using a numeric qualifier or by adding or replacing the last part of the name by a numerical value. For data sets with otherwise identical names, this number indicates the delivery sequence of the data sets (unless one or more old ones have been deleted, rvs® uses the smallest available number). The file systems currently supported by portable rvs® do not allow addition of a real time stamping, i.e. date and time of delivery, to the file name.

Time stamping may be requested in a resident receive entry. It is done, when the data set is delivered.

2.13. Memory line traces

rvs® offers the possibility of line traces in memory from the version 4.0 above. This feature is very important for diagnosing line (network) errors, which occur now and then.

All incoming and outgoing data will be logged in memory.

The size of the memory buffer can be configurated by the new rvs® parameters `TRACEMAXITEM` and `TRACEMAXSIZE`.

`TRACEMAXITEM` is the number of records in memory and `TRACEMAXSIZE` is the maximum size of the traces in memory (unit: kB). The default value for `TRACEMAXITEM` is 1024 records and the default value for `TRACEMAXSIZE` is 1024 kB.

If you want to store the all line traces in a file, you should configure the new parameter TRACEALWAYSTOFILE (=1). The severe line errors will always be traced into a file (also without the configuration of the parameter TRACEALWAYSTOFILE).

The trace files will be stored in the directory $RVSPATH/tracedir under the name traceXXX.log. (where XXX is the number of the rvscom process ID; this ID is also to find in the rvs MonitorLog file $RVSPATH/system/rlog.log).

The parameter TRACECONNERR (if it is active: =1) causes that each connection error will be written into a trace file $RVSPATH/tracedir/traceXXX.log.

2.14. The robustness of rvs® concerning disk space

rvs® from the version 4.05 enables the possibility of disk space checking of all important rvs® directories. This increases the robustness of rvs® concerning disk space problems.

This functionality allows the operator to react early enough to lack of free disk space. In case of low disk space you can see in the Monitor Log a message, that informs you in which rvs® directories is not enough disk space. Depending on the degree of disk space lackness you have the possibility to dispay a warning or an error message. After an error rvs® will be stopped.

Example of a warning message:

```
W:  <DISKSPACEWARN   >Diskspace an /home/uitt/rvs/db, /home/uitt/rvs/temp
, /home/uitt/rvs/tracedir , /home/uitt/rvs/usrdat/ ,
/home/uitt/rvs/temp/in ,
 /home/uitt/rvs/temp/out , /home/uitt/rvs/temp/temp low, needed  11021504
kb , available 8610144 kb.
```

Example of an error message:

```
W:  <DISKSPACEERR    >Diskspace an /home/uitt/rvs/db ,
/home/uitt/rvs/temp , /home/uitt/rvs/tracedir , /home/uitt/rvs/usrdat/ ,
/home/uitt/rvs/temp/in , /home/uitt/rvs/temp/out ,
/home/uitt/rvs/temp/temp extrem low, needed  11021504 kb , availiable
8609792 kb.
```

After an error rvs® will be stopped and the following message will be dispayed:

```
A:  <OK_READ         >  [rvsstop] stop rvs=force
I:  <OK_CMD_DONE     >  [rvsstop]  'stop' done.
I:  <XMT_STOP_NORMAL >  Master Transmitter stops normal.
I:  <RVS_TERMINATION >  RVS Monitor terminates. (return code: 4).
```

In the case of a warning the script `diskspacewarn` will be called. In the case of an error another script with the name `diskspaceerr` will be called. The both scripts are examples from the directory `$RVSPATH/system` and can be adjusted to your needs. These scripts must be stored in the directory `$RVSPATH/system` and also named exactly as mentioned above.

Note: Scripts for UNIX platforms end with `.sh` (shell files) and scripts for windows end with `.bat`.

Example (`diskspaceerr.sh`):

```
echo "1 >$1<  2 >$2< 3 >$3<" >> $HOME/diskerr.log
echo "DISKSPACEERR : low disk space on >$1< ( needed $2 kb,
available $3 kb)" >> $HOME/diskerr.log
rvsstop -f
```

In this example script the log message will be additionally logged in a file diskerr.log and immediately afterwards rvs® will be stopped.

How to define a warning (how much of free disk space in rvs® directories should be available) can be configured by the parameter `DISKSPACEWARN` (see the table at the end of this chapter). An error is to be defined by the parameter `DISKSPACERR`.

The following rvs® directories will be regulary checked for free disk space: `DB`, `TEMP`, `TRACEDIR`, `USRDAT`, `SPINDIR`, `SPOUTDIR` und `SPFILEDIR`. The time interval for the checking should be configurated by the parameter `DISKSPACEDELAY` (see the table at the end of this chapter).

If the free disk space in the directory `$RVSPATH/temp` is low for the receipt of a file, the message "file too big" will appear in the Monitor Log. How much of free disk space has to be available for the successful receipt is to be configured with the parameter `DISKSPACEEJECT`.

Example (Monitor Log with the message "`file too big`" on the sender side):

```
A: 2006/03/15 14:15:19  <NEW_CMD_CREATED >  SK(125) created by SE(124).
A: 2006/03/15 14:16:26  <SENDER_STARTED  >  SK(125) Sender started to
SID(WOB_AIX) (using Prot(TCP/IP)).
A: 2006/03/15 14:16:26  <CALL_OUT        >  1196: outgoing call to
SID=WOB_AIX ...
O: 2006/03/15 14:16:27  <CONNECT         >  Sender: Connection with
Station 'WOB_AIX' with Credit=99, Odette Buffer=2000, OFTP compression
established.
```

```
O: 2006/03/15 14:16:27  <OFTP_SEND       >  Send:SK 125 'TESTTEST(060315
141519)' from:'LOC' Destination:'WOB_AIX' FORMAT=U RESTART=0.
A: 2006/03/15 14:16:27  <RPS_TERMINATION >  SK(125) Sending ended code 4
   (transmission rejected by neighbor (SFNA or EFNA))
O: 2006/03/15 14:16:28  <DISCONNECT      >  Connection as (send) to
Station 'WOB_AIX' ended.
I: 2006/03/15 14:16:51  <SERR_ODETTE     >  rpm: Node WOB_AIX rejected
Odette transfer for SK(125).
       Send File Negative Answer - retry allowed : file size is too big
```

Example (Monitor Log with the message “`file too big`” on the receiver side):

```
I:  <INCOMING_CALL   >  Incoming call received: DIEXP
O:  <CONNECT_IND     >  Responder: Connection with Station 'DIEXP' with
Credit=9
9, Odette Buffer=2000, OFTP compression established.
E:  <SHOW_CAUSE      >Error occured when starting or ending file transfer
(CAUSE
=6):  SFNA: file too big
O:  <DISCONNECT      >  Connection as (receive) to Station 'DIEXP' ended.
```

Here is the overview of the new rvs® parameters concerning the robustness of disk space:

Parameter	
DISKSPACEERR	Number of Kilo Bytes, that has to be free in every rvs® directory, which has to be checked. If the number of free Kilo Bytes is less than a value of this parameter the script `diskspaceerr` will be called. Default: 1000 (ca. 1 MByte)
DISKSPACEWARN	Number of Kilo Bytes, that has to be free in every rvs® directory, which has to be checked. If the number of free Kilo Bytes is less than a value of this parameter the script `diskspacewarn` will be called. Default: 5000 (ca. 5 MByte)
DISKSPACEDELAY	Time interval in seconds for the apperance of `DISKSPACEWARN` in the Monitor Log and the call of the script `DISKSPACEWARN`. Default: 600 seconds (10 minutes)
DISKSPACEREJECT	Number of Kilo Bytes, to be free in the directory `$RVSPATH/temp` for the receipt of files. Default: 5000 (ca. 5 MByte)

2.15. rvs®-PKI-Binding

In a PKI (Public Key Infrastructure) are stored public keys with their certificates. The access to the PKI takes place via LDAP.

Note: LDAP is a network protocol, which manages the communication between a LDAP Client and a LDAP Directory Server. This protocol offers the following features: login from client to the server, search queries for the information stored in the directory and if necessary: modification of information. This means relating to the PKI: in the LDAP Directory Server the public keys and certificates are administrated.

Following features are available for rvs®-PKI-Binding:

- Access of public keys for partner station
- Check of public keys with OCSP (Online Certificate Status Protocol)
- Transfer of information from a OCSP server with HTTP

The rvs®-PKI-Binding will be activated with the rvs® parameter `USEPKI`. This parameter kann be used related to a station (in the ODETTE station table – OP) or global for all station as an rvs® global parameter.

rvsX

Global als rvsX-Parameter for all stations in the `$RVSPATH/init/rdmini.dat` file.

```
setparm USEPKI=Y
```

rvsXP

Related to a station: `rvsXP Administrator -> Stationen -> ODETTE Parameter -> PKI`

Global: `rvsXP Administrator -> Parameter -> USEPKI`

The configuration of the PKI-Binding takes place in a file `$RVSPATH/init/rvspki.dat`.

If you have activated the parameter `USEPKI (Y=YES)`, the parameter for the access to a PKI should be configurated additionally in the file `$RVSPATH/init/rvspki.dat`.

Example ($RVSPATH/init rvspki.dat):

```
[PKI]
#Host name or IP address of LDAP directory server
DirectoryServer=17.145.32.89

#IP port of LDAP directory server, 389 is used as default, if
left empty
DirectoryServerPort=

#User name and password of LDAP directory server,
#anonymous access is used if left empty
DirectoryServerUser=
DirectoryServerPw=

#LDAP version of LDAP directory server, 3 is used as default, if
left empty
DirectoryServerLDAPVer=

#dn of LDAP search starting point, from special to global
property
LDAPSearchBase=ou=Odette,o=BTGK,dc=BT,dc=hgf,dc=com

#LDAP search filter part 1, Odette ID of the partner station is
part 2
LDAPFilterPart1=cn=BT_OFTP

#URL of OCSP server
OCSPServerURL=http://ocsp.btgk.de

#Options
#Skip OCSP check: Options=1
#Ignore OCSP check result and errors: Options=2
Options=2

#Location and file name of the private key file
PrivateKey=/rvs/init/btgk.pri

#Service entries for support purposes only
#PKIStr1=
#PKIStr2=
TestMode=0

[Issuer]
#Locations and file names of certificate issuers' certificates
<CA #1 name>=<path and file name>
<CA #n name>=<path and file name>
```

The mandatory parameters are: `DirectoryServer`, `LDAPSearchBase`, `LDAPFilterPart1` und `PrivateKey`.

In the following table are described the parameter from `rvspki.dat` file:

Parameter	Meaning
`DirectoryServer`	Host name or IP address of the LDAP Directory Server
`DirectoryServerPort`	Port of the LDAP DirectoryServer Default: 389
`DirectoryServerUser`	User name in LDAP-DirectoryServer
`DirectoryServerPw`	Password of the user in LDAP-DirectoryServer
`DirectoryServerLDAPVer`	Version of LDAP protocol in LDAP DirectoryServer Default: 3
`LDAPSearchBase`	Describes the structure in the directory, where the search should begin. `ou` means OrganizationalUnit, o Organization and `dc` DomainComponent.
`LDAPFilterPart1`	LDAP search filter; Example: for the VW PKI binding ist the first part of the filter `cn=VW_OFTP`; the second is the OdetteID of the appropriate station.
`OCSPServerURL`	Here should be defined the URL of the OCSP Server. OCSP (Online Certifikate Status Protocol) is used for obtaining status of digital certificates. It will be checked, if a certificate is unknown, valid or disabled.

`Options`	This parameter defines the OCSP check. Possible values: • 0: OCSP check will not be ignored. • 1: OCSP check will be ignored. • 2: OCSP errors will be logged, but no abort will be caused. It is recommended Options=2; Options=0 can mean, that for the time that OCSP server can not be reached, the encryption and decryption do not work.
`PrivateKey`	Path and the file name of the private key
`ServiceEntries (TestMode)`	Only for test purpose with T-Systems customer service
`[Issuer]: Location and file names of certificate issuers' certificates`	Path and the file name of the certificate issues' certificates

3. Protocol Layers in rvs® Communication

General Overview

The functional hierarchy of protocol layers in a rvs® environment can be regarded as sub layers of the application layer (layer 7) of the ISO/OSI Reference Model. The rvs® linedriver and the communication system components it builts on, like X.25 native, SNA LU6.2 or TCP/IP, belong to the lower levels. However, rvs® is not compatible with ISO/OSI standards, it rather follows the recommendations of the ODETTE group. The following gives a simplified view of the functional hierarchy, for example with LU 6.2 communication:

```
          LOCAL SYSTEM                         REMOTE SYSTEM
User, Operator, Programs, Jobs     User, Operator, Programs, Jobs
  0  !                                               !
  1  !                                               !
   MONITOR → local Job submit         MONITOR → local Job submit
  2  !                                               !
     MASTERTRANSMITTER                   MASTERTRANSMITTER
  3  !                                               !
(local) Data ↔ COMMUN. PGM.      (local) Data ↔ COMMUN. PGM.
  4  !                                               !
        OFTP         -------OFTPsession-------      OFTP
         !                                           !
     LINEDRIVER      -----linedriver session ---  LINEDRIVER
         !             (LU 6.2 conversation)         !
         !                                           !
      NETWORK        ------Network session ----    NETWORK
         !                 (SNA session)             !
         !                                           !
           -------------------physical network--------------------
```

3.1. Network

rvs® does not care much about the physical details of the network, e.g. whether X.25, leased lines, or Token Rings are used. The control and definitions of the physical network is external to portable rvs®. For SNA networks, the availability of LU 6.2 services and PU 2.1 support for both communication partners is mandatory.

3.2. Linedriver

The rvs® linedriver together with the related system components is used to establish an end-to-end connection (X.25 connection, LU 6.2 conversation or TCP/IP connection) which allows to transport data packages transparently and reliably from one rvs® node to another. It talks to the remote rvs® linedriver on the basis of a special linedriver protocol which depends on the network type, e.g. X.25, LU 6.2 or TCP/IP. The rvs® linedriver receives network connect or disconnect requests from the ODETTE File Transfer Protocol (OFTP) layer. After successful connection, it receives data packages from the remote linedriver and delivers them transparently to the local OFTP layer and vice versa.

3.3. OFTP

It is the purpose of the ODETTE File Transfer Protocol (OFTP) to ensure the reliable transfer of a data set. The OFTP enters a protocol session with the OFTP on the remote rvs® station which logically runs on top of the linedriver connection.

After the OFTP session has started, both sides exchange their ODETTE IDs and passwords, negotiate some parameters, like ODETTE exchange buffer size, ODETTE credit value (the number of buffers the sending side can send without waiting for a response), and exchange information about name, approximate size and format of the data set to be transferred.

During transfer, a compression and decompression of data is performed. After the data have been transferred, the byte count is checked between both sides. After the data set has successfully been stored, a receipt is sent to the sending station. If the transfer has been disrupted, for example by a link failure, the OFTP protocol provides a mechanism that allows to restart the transfer at the point of rupture.

'Change direction' feature: When all Data sets are transmitted, the sender asks the receiver to send its queued data sets.

For protocol details kindly refer to the publications of the ODETTE and VDA groups: "ODETTE Specifications for File Transfer".

Shown below is the general but simplified message flow within an ODETTE session. The sending side acts as initiator, the receiving side as responder.

Initiator (Sender)		Responder (Receiver)
Connect	––– network connect –→	
	←––––––– SSRM ––––––	"ODETTE FTP READY"
ODETTE id password buffersize, credit	––––––– SSID ––––––→	
	←–––––––– SSID ––––––	ODETTE-id password etc.
Start File-id (name, size, format)	––––––– SFID ––––––→	
	←––––––– SFPA ––––––	Start file pos. answer
'n' data records	======= DATA ======>	
	======= DATA ======>	
	. . .	
	======= DATA ======>	
	←––––– CREDIT –––––	send credit value 'n'
send 'n' records	======= DATA ======>	
	.	
	.	
	.	
end file identif. (bytecount)	–––––––– EFID –––––→	
	←––––––– EFPA ––––––	end file positive answer (if store successful and bytecount correct)

change direction	––––––––– CD ––––––→	
	←––––– EERP ––––––	end to end response (receipt)
	←––––––– ESID ––––––	end of session identif.
network disconnect		network disconnect

3.4. Communication Program

The rvs® communication program handles the system dependent physical file I/O on top of OFTP. If necessary, the data are translated from EBCDIC to ASCII code or vice versa. The translation table is a separate data set, that can be modified by an installation to meet some special requirements. The Communication program, when acting in receive mode, stores the incoming data into a temporary data set. Even data formats that are not supported on the local system, like for example fixed or variable record files under OS/2, can be stored and forewarded; they will be converted to a supported format upon delivery.

4. LU 6.2 Basic Concepts

The purpose of this chapter is to provide a brief overview over the basic functionality of LU 6.2 communication and to facilitate understanding of the correspondent rvs® stationtable parameters. Implementation details and limitations as they exist for certain system platforms are mentioned in the respective installation manuals. If you need details about LU 6.2, kindly refer to the respective publications from IBM and/or the LU 6.2 documentation of your system supplier. Some useful IBM publications are listed below:

- Systems Application Architecture, Communications Reference, SC26-4399
- System Network Architecture, Concepts and Products, GC30-3072-4
- System Network Architecture, Sessions Between Logical Units, GC20-1868-2
- System Network Architecture, Type 2.1 Node Reference, SC30-3422-2
- Overview sections in system specific APPC manuals, like

– OS/2 (EE 1.3) APPC Programming Reference
– AS/400 APPC Programmer's Guide
– RS/6000 APPC Programmer's Guide

4.1. Why LU 6.2 Communications?

The most important reasons for choosing the LU 6.2 (SNA Logical Unit type 6.2) communications technique as one future main path for all rvs® versions are:

- An Advanced Program to Program Communication (APPC) application interface for LU 6.2 is available on almost every state of the art computer system.
- The APPC programming interface is standardized, at least on a functional level. Thus, application programs using APPC can be adapted to other systems without major redesign.
- The application interface is independent of the physical properties of the network. The application just picks up an available SNA session for use.
- The control of network resources out of the application program is normally not necessary and often not even possible with APPC. This has to be done externally using the respective systems network operator tools.

The alternative within the SNA world would be to use the outdated LU 0 interface. LU 0 is not supported on many computer systems; and if available, the programming interface is system dependent to a large extent. In addition, it requires some effort for network control.

4.2. LU 6.2 Basic Functionality

The SNA network and the APPC application interface must be seen as almost independent issues. In the framework of the OSI 7-layer model, APPC belongs to layer 7, while the network itself covers the layers 0 to 6 (roughly speaking).

The SNA network must provide a physical and logical path between the communication endpoints, the type 6.2 LUs, which usually are real computer systems that act as equivalent peers in the network. This path is an SNA session established with a special SNA logmode. Details of the physical network, e.g. wether Token-Ring, X.25, SDLC links are used, must be dealt with on the level of definitions in the respective basic communication resources like VTAM, NCP, etc..

The LU 6.2 application, in our case rvs®, just uses an available SNA-session in order to setup a logical communication path between the partners. This logical path is called an LU 6.2 conversation. A conversation can only exist between a pair of application programs. One basic feature of LU 6.2 is, that when the local program initiates a conversation, it starts up a partner program on the remote site. The local application must only know the LU-Name of the partner system, the name of the (Log)Mode to select an unused session with the right characteristics, and the name of the remote partner transaction program (TP-name).

In our case the TP name is that of the rvs® communication process: `rvscom`. After the conversation has been built, both partners communicate by means of APPC verbs which are more or less standardized function calls like for example "Send" or "ReceiveAndWait". LU 6.2 conversations have half-duplex flip-flop characteristics_therefore, only one side at a time has control.

4.3. Mapped or Basic Conversation

Depending on initialization parameters, a conversation can be of type "mapped" or "basic" with synchronization levels of "confirm" or "none".

The main difference between basic and mapped conversation is, that in basic mode, the length information about the block of user data to be transferred is part of the data. The application has to insert the length information into the data when sending and retrieve it from the data when receiving. Presently, rvs® supports only mapped conversations, where the system accepts and returns the length by means of a parameter of the send or receive function calls.

Synchronization level "confirm" means, that for synchronization purposes one partner can request the other to send a confirmation signal and the conversation will not proceed until the confirmation is sent. rvs® supports either mode. Using synclevel "confirm" is more secure but produces more overhead.

4.4. Security

Most systems provide the possibility to transmit security information along with the "allocate" request, which is used to initialize a conversation. This security information consists of a user ID and a password valid for the remote system; they are necessary to start the remote transaction program. Some systems support default users and do not check security when such a default user has been defined. rvs® provides correspondent stationtable parameters for security information.

4.5. Dependent and Independent Lus

There are two types of LU 6.2 logical units: independent LUs (real peers) and dependent LUs. The latter are LUs that are controlled from a system services control point, SSCP (e.g. VTAM on an SNA host). Dependent LUs are treated as local resources (like a terminal) belonging to a physical unit of type 2.0, (PU 2.0).

Independent or peer LUs require support of physical units of type 2.1, (PU 2.1). On non-mainframe systems, the independent LU and the PU 2.1 is the system itself. Independent LUs and the related PUs of type 2.1 are always considered as "active" and never accept SSCP commands such as "actpu" or "actlu" from outside. Consequently, they also cannot be deactivated from outside. Before a SNA session is established, i.e. a bind image is exchanged, both sides exchange their XIDs. This is done even when the connection is based on leased lines, because the XID-exchange is used to negotiate the station roles (primary or secondary) of either partner.

rvs® is designed to be used with independent LUs and PU 2.1. One basic advantage of a PU 2.1 is, that parallel sessions are supported. This means, that all partner LUs can freely access the local LU or freely be accessed by the local LU at the same time. With independent LUs, rvs® can open many conversations simultaneously and transfer data in parallel on the same physical link.

The use of dependent LUs has not yet been tested with rvs® and therefore is declared as "not yet supported". For dependent LUs, the session initialisation is controlled by an SNA-host, and only a single session per LU is possible. On such a connection, rvs® could transfer only one data set at a time.

While most non-mainframe systems support PU 2.1 as easy-to-use standard, some effort and knowledge is required in order to provide PU 2.1 support on SNA-hosts.

4.6. Effects of LU 6.2 on rvs® Design

Because of the basic LU 6.2 characteristics, rvs® had to be designed in a somewhat special way, which is different from the rvsMVS implementation. On non-mainframe systems, the LU always is the system itself and the TP the rvs® Receiver in our case is started as an independent main process.

Some LU 6.2 implementations (for example the one under SINIX) even require that every APPC application must run as independent main task. For symmetry and portability reasons, both, Sender and Receiver have been consolidated into a unique Communications program that can assume either role. The Communications program was designed as independent job or process.

The rvs® MasterTransmitter has the role to submit or initiate an independent Communications processes whenever a data set has to be transferred. MasterTransmitter and Communications processes communicate via system dependent semaphores and via the rvs® database.

This is different from the implementation of rvsMVS where all LU 6.2 tasks run as subtasks of the Monitor. In rvsMVS the LU is the rvs® subtask for LU 6.2 control and the TP is a subtask of that subtask.

5. X.25 Native Communication

General X.25 Characteristics

The pupose of this chapter is to provide a general overview over X.25 communications technics per se and their usage with rvs®. The implementation differences and limitations for special system platforms are dealt with in the respective installation manuals.

X.25 native communication is the basis for `open' communication as defined in the standards of the ODETTE file transfer protocol. The meaning of `open' is world wide connectivity between nodes running an OFTP compatible communcation product like rvs® via public packet switched data networks. X.25 networks are based on ISO/OSI definitions and cover OSI-layers 1 thru 3. Any installation that is attached to a public X.25 network can be reached or addressed by a unique X.25 call address of up to 15 decimal digits. Thus an almost telefone-like connectivity is achieved. However, not all national x.25 networks are functional on the same level and limitations may occur.

In an X.25 network the physical routing and the transport of data, which are grouped into data packets of 128 bytes, is performed by the public network supplier. It is possible that data packets belonging to the same block of user data travel different routes, however the network will deliver them in sequence at the destination. The network also provides an automatic speed control, thus allowing communication between partners using different transmission speeds. Special X.25 protocol elements (RR, RNR) provide an end-to-end pacing control to prevent data overruns.

The end-to-end connection in an X.25 network is called a virtual circuit, VC. One physical link to the X.25 network can carry up to 255 VCs, depending on what you have ordered from your network supplier. Because of this capabilities the link into the packet switched network is often called a multi channel link.

There are two types of VCs:

1. SVCs, switched virtual circuits,

2. PVCs, permanent virtual circuits.

PVCs behave like a leased line and represent a distinct point-to-point relationship. SVCs have a point-to-point character too, but the other end can be freely choosen in the initialisation phase. At the end of a logical connection the SVC is disconnected. rvs® is designed to operate on the basis of SVCs.

Customers attaching to an X.25 network will have to specify how many PVCs and/or SVCs they want to have configured for their "multi channel". The number of channels is an upper limit to possible parallel connections. On the other hand, the number of virtual circuits is a cost factor. The requirements of rvs® concerning number of VCs depend on the expected data traffic, precisely, on the number of data sets that must be transferred in parallel at a time.

As a minimum requirement for very low traffic, 2 VCs should be available. In normal cases any number around 10 VCs should be sufficient.

X.25 networks, dependent on supplier, offer a variety of additional features like

- reverse charging
- closed user groups
- fast select
- delivery confirmation
- call user data

and many others. Most of them must be mutually agreed upon between partners.

Most X.25 networks support a feature called Packet Assembler Disassembler, PAD. A PAD is a hard- or software component, which converts simple standard ASCII data streams into X.25 packets and vice versa. A PAD can only be called from the ASCII side. On the X.25 side, it only can call out. A standard PAD configuration is an ASCII-terminal or PC calling a public PAD via telefone modem, then accesses a remote Host via public X.25 network.

The standard protocol for the ASCII side is X.28, the user configurable PAD parameters are standardized as X.3 and the end-to-end connection protocol in this context is standardized as X.29. The ability to support PAD connections will be provided in one of the next portable rvs® versions.

Instead of using an expensive public X.25 network, like DATEX-P, it is possible to install an ISDN adapter between X.25 board and an ISDN port. The adapter translates X.25 packets to ISDN and vice versa. This protocol is standardized as X.31. Both communication partners need an adapter. If you have questions, please ask your distributor.

5.1. Connection Establishment

The addressing of a partner rvs® via X.25 requires basically the X.25 call address and optional X.25 call user data.

The connection establishment on X.25 network level begins with the send-out of a X.25 Call Packet consisting of the address to be called, the calling (own) address followed by call user data. If the remote side sends back the call data packet, the connection is established and an SVC activated. If the remote station does not expect the call or if any of the network parameters is odd, it will answer the call with a Clear packet which also terminates the SVC.

Successful connection establishment requires that a "listening" rvs® task is active on the side receiving the call. On some systems like IBM-Mainframes, rvs® activates a responder task upon receipt of an X.25 call. On others, like most portable rvs® platforms, one ore more active receiver task must be prestarted in order to catch incoming calls.

Once an SVC is established, both sides can freely exchange data buffers in full-duplex mode. The only security check so far is that the called station examins the callers address. For better security, the ODETTE FTP used by rvs®, requires the exchange of SSIDs with station ID's and passwords. If both sides agree, the ODETTE session is established and file transfer can start.

Under normal conditions, rvs® does not require any call user data. However, if other applications share the same X.25 multichannel, the call user data must be used to route an incoming call to the correct application. The first byte of call user data is usually interpreted as so-called protocol identifier, pid. Some characters are reserved as pid: X'C3 and X'C4 for SNA, X'01 for PAD, X'C0 for ASYNC etc. By default rvs® expects no PID at all. PIDs of X'01 are encountered in calls originating from PADs and X'C0 in calls originating from some less sophisticated AS/400 products. Those PIDs can be of special importance for rvs®.

5.2. Effects of X.25 on rvs® Design

With X.25 native communication, the need for better control of the number of simultaniously active transmitters became obvious, since customers may use X.25 multi channels with only very few virtual circuits because of cost reasons. As a result, the MasterTransmitter was introduced into rvs®. In order to be able to deal with X.28 communication partners dialing in via an X.25 PAD, the formerly separated sender-only and receiver-only modules were integrated into a unique Communications Program that can handle both directions. This is necessary because PAD Connections always have only one single "session" which in addition can only be initiated from the X.28 side.

rvs® supports multiple X.25 lines. If more than one X.25 line is used, rvs® needs a database entry ('XP') for every line (with adress, link or alias name etc.).

6. TCP/IP and rvs®

This chapter describes the TCP/IP application interface, the TCP/IP addressing as well as the establishing of the connection with rvs®.

6.1. TCP/IP Application Interface

rvs® does not care how and which route within a TCP/IP network data are transported. It rather uses an application interface on a fairly high level, consisting of not much more than the basic function calls "connect", "listen", "accept", "send", "receive" and "close" of the commonly available C-socket library. The C-socket library allows for reliable transmission of blocks of data.

Communication via TCP/IP requires that the partner who takes the initiative to set up a connection acts as a "client" while the counterpart must act as a "server". The client process, in our case the rvs® Sender, issues a connect request to an address where it knows that the partner, the rvs® Receiver, is `listening' as a server process. The connection is established and ready for use when the server process has issued an accept call which will succeed when the connect request from the client arrives. From then on, full duplex exchange of data is possible.

6.2. TCP/IP Addressing

Addressing in an TCP/IP network is fairly simple. A TCP/IP address consists of two parts:

Internet address	This is a world-wide unique address which defines a host system. It is a 32-bit integer value, sometimes given in the form "aaa.bbb.ccc.ddd", where `aaa' to `ddd' are 3-digit decimal values each describing one of the 4 bytes of the Internet address from left (high order) to right (low order).
Port number	Each host in the network can use a range of 64 K subaddresses called ports. Each port is described by an 16 bit integer value. Port numbers 0 to 1023 are reserved.

A TCP/IP connection is always established between exactly one local and one remote internet.port combination, where one side must act as client, the other as server. An internet.port combination is bound to a so called socket, which the application program internally uses as reference. While the client is free to choose any free port, the server is not. The server must be listening on one destinct port which must be known to the client; otherwise, the client will not be able to find the server and may even unadvertantly attach to the wrong application.

6.3. Establishing a Connection with rvs®

rvs® is designed to run parallel transmissions on parallel TCP/IP connections.

One possibility is to provide a range of port numbers on each system where rvs® receivers could be listening as servers. The problem is than that this range must be coordinated and communicated througout all other rvs® stations, which seemed not to be very practicable.

Instead the following dynamic two-stage addressing concept was developed.

Each rvs® on the TCP/IP network runs a permanently listening master server task which is waiting on exactly one fixed portnumber which has to be known by all other rvs® systems.

Whenever a sender task is started at an originator site, it first connects to the master server port at the destination site. The necessary address information is easily derived from the stationtable. After the connection to the master server is established, the master server communicates on this connection the port address of a prestarted receiver task. The connection is terminated and the sender task on the originator site reconnects to the destination site with the new port number behind which the rvs® receiver is waiting for the work connection.

The master server meanwhile prestarts another receiver on any random port which he waits to communicate as work port to the next client who calls in.

7. XOT Router Configuration

In this chapter we will give some recommendations for the configuration of XOT routers.

XOT (X.25 over TCP/IP) routers are able to route X.25 packets between a TCP/IP network on one side and an X.25 or ISDN network on the other side. Please read the User Manual for details how to configure rvs® for XOT.

7.1. System requirements

To use the XOT functionality in rvs® you need an IP connection to the XOT-capable router (e.g. CISCO 801, CISCO 2600 or BINTEC X4300).

XOT as a new network type in rvs® is at the moment available for the following platforms:

- LINUX
- Windows.

We tested rvs® and XOT with the following routers:

- CISCO 801
- CISCO 2600 and
- BINTEC X4300.

7.2. CISCO Configuration

In the following chapter we will describe our test scenario and give some examples for CISCO router configuration.

To configure one single BRI interface for incoming and outgoing direction we use dialer interfaces. There is at least one dialer for the direction „send“ and at least another dialer for the direction „receive“. Before we tested the CISCO router with rvs® and XOT, we checked the X.25 function with PAD commands directly on CISCO-IOS.

7.2.1. CISCO 801

We used firmware 12.3. We had some trouble with firmware 12.2.

7.2.2. CISCO 2600

We used firmware 12.2. We had some trouble with firmware 12.0.

7.2.3. Links for more information

For further information about XOT and CISCO see CISCO manuals. Here some interesting links:

http://www.cisco.com/en/US/tech/tk713/tk730/tech_digests_list.html
http://www.cisco.com/en/US/products/sw/iosswrel/ps1818/products_configuration_guide_chapter09186a0080087858.html
http://www.cisco.com/en/US/tech/tk713/tk730/technologies_q_and_a_item09186a00800a3c0b.shtml
http://www.cisco.com/en/US/products/hw/routers/ps380/tsd_products_support_series_home.html
http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122relnt/800/rn800xi.htm#1075191
http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123relnt/800/rn800xa.htm

7.2.4. Examples of CISCO Router Configuration

In this chapter we'll give some advices for configuration of CISCO router:

- You must configure the outgoing dialer interface with **encapsulation x25 DTE**.
- The incoming dialer interface must be configured with **encapsulation x25 DCE**.
- There is an idle-Timer for incoming ISDN-connections. This timer does not recognize x25 packets over ISDN, so it is possible that the connection will hang up during transmission. You should set this timer in the incoming BRI interface to a big value, e.g. **dialer idle-timeout 700000.**
- You have to configure the routing for both directions.
  Every outgoing dialer represents a partner station with its own ISDN number. (dialer string <ISDN-No>). The incoming dialer listens to the own ISDN numbers. (dialer called <ISDN-No>).
- You can route incoming XOT calls with the following routing-entry: **x25 route [called x25-Address in the XOT-packet] interface DIALER[number]** .
- You can route incoming ISDN/X.25 calls with the following routing-entry:
  x25 route [called x25-Address in the x25-packet] XOT [IP of the XOT-station] .

Here are some configuration examples for CISCO routers:

Example: CISCO 801

```
version 12.3
service pad to-xot
service pad from-xot
service tcp-keepalives-in
service tcp-keepalives-out
!
hostname Receiver
!
isdn switch-type basic-net3
x25 routing
!
interface Ethernet0
 ip address [IP-ADDRESS]
 no cdp enable
!
interface BRI0
 no ip address
 encapsulation hdlc
 no ip mroute-cache
 dialer pool-member 1
 dialer idle-timeout 1
 isdn switch-type basic-net3
 isdn point-to-point-setup
 no fair-queue
 no cdp enable
!

interface Dialer1
 description outgoing ISDN calls
 no ip address
 encapsulation x25
 no ip mroute-cache
 dialer pool 1
 dialer idle-timeout 1
 dialer string <partner-ISDN-number 1>
 dialer max-call 1
 dialer-group 1
 x25 htc 1
 x25 win 7
 x25 wout 7
 no cdp enable


interface Dialer10
 description incoming ISDN calls
 no ip address
 encapsulation x25 dce
 no ip mroute-cache
 dialer pool 1
 dialer idle-timeout 700000
 dialer called <own ISDN number 1>
 dialer called <own ISDN number 2>
 dialer max-call 1
 dialer-group 1
 x25 htc 1
 x25 win 7
 x25 wout 7
 no cdp enable

x25 route [X.25-ADDRESS-1] xot [IP-ADDRESS-RVS]
x25 route [X.25-ADDRESS-2] xot [IP-ADDRESS-RVS]
x25 route [X.25-ADDRESS-1] interface Dialer1

x25 host name [X.25-ADDRESS-ROUTER]
```

Example of CISCO Router Configuration (serial)

```
service pad to-xot
service pad from-xot
service tcp-keepalives-in
service tcp-keepalives-out
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname C2600
!
!
ip subnet-zero
isdn switch-type basic-net3
x25 routing
!
!
!
!
!
interface Ethernet0/0
 ip address 192.168.2.4 255.255.252.0
 no ip directed-broadcast
 no cdp enable
!
interface Serial0/0
 no ip address
 no ip directed-broadcast
 shutdown
!
ip classless
!
no cdp run
!
x25 route input-interface Serial0 xot 192.168.2.10
x25 route ^. interface Serial0/0
x25 host Router name1
!
line con 0
line aux 0
line vty 0 4
 login
!
end
```

Example CISCO Router 2800 for X.25

```
interface Serial0/0/0
 no ip address
 encapsulation x25
 no ip route-cache
 no ip mroute-cache
 x25 htc 15
 clock rate 2000000
 no cdp enable
!
x25 route ^. interface Serial0/0/0
```

7.3. BINTEC X4300 Configuration

In this chapter we will give an example for the BINTEC X4300 router configuration.

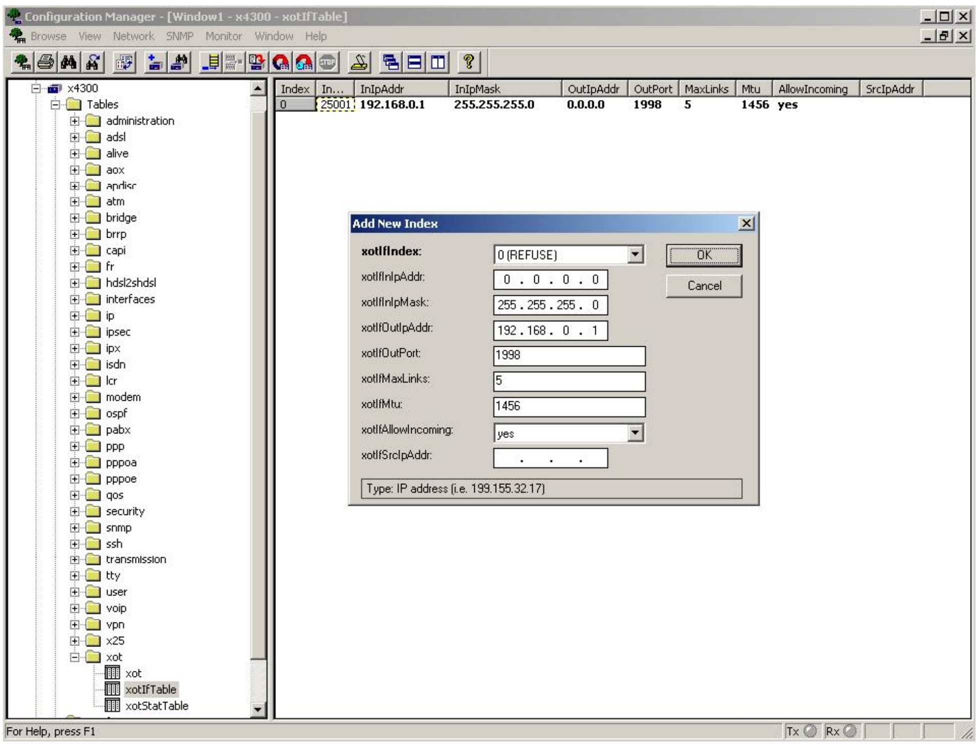
In out tests we used BINTEC X4300 with firmware version 7.1 Rev. 1.

More information about the BINTEC X4300 you can find here:

http://www.funkwerk-ec.com/dl_bintec_x4x00_family_en,17190,837.html

7.3.1. Configuration of the BINTEC router X4300 for XOT and XOI

1. Define an XOT Interface (*xotIfTable*)
   - You have to specify one interface for every communication-partner and direction (incoming/outgoing)
   - For an incoming XOT connection you have to set the parameter `xotIfOutIpAddr` to 0.0.0.0 and the parameter `xotIfInIpAddr` to the IP address of the partner station.
   - For an outgoing XOT connection you have to set the parameter `xotIfInIpAddr` to 0.0.0.0 and the parameter `xotIfOutIpAddr` to the IP address of the partner station.



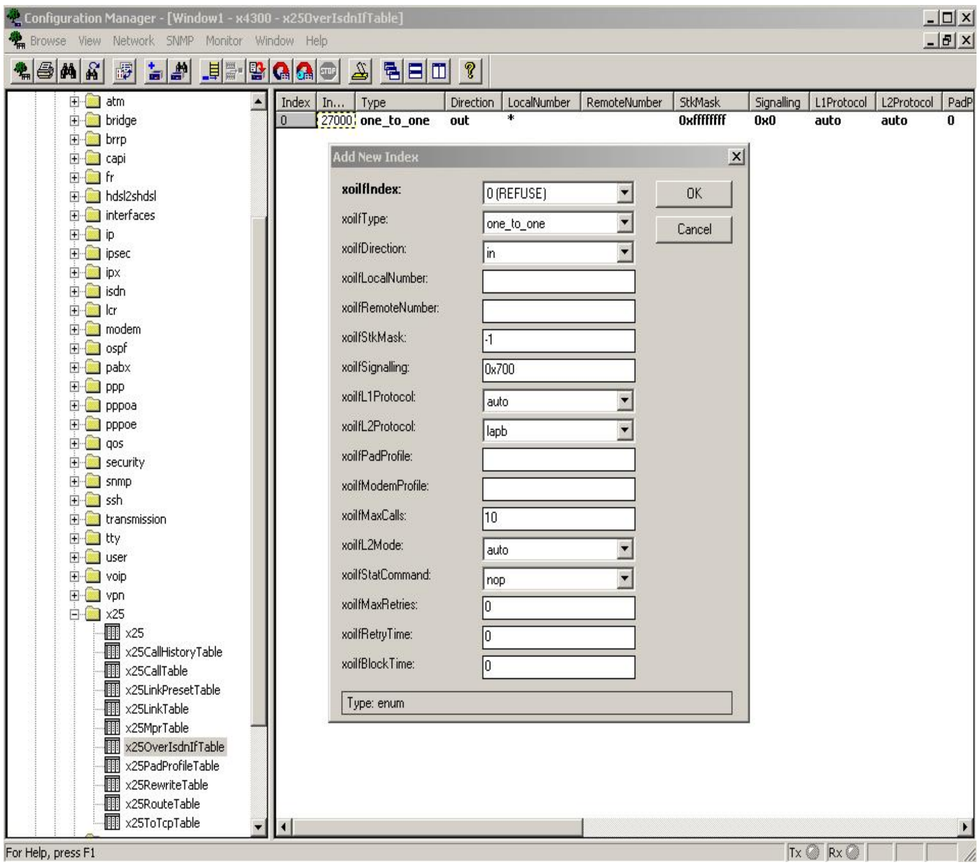
Index	In...	InIpAddr	InIpMask	OutIpAddr	OutPort	MaxLinks	Mtu	AllowIncoming	SrcIpAddr
0	25001	192.168.0.1	255.255.255.0	0.0.0.0	1998	5	1456	yes	

xotIfIndex:	0 (REFUSE)
xotIfInIpAddr:	0 . 0 . 0 . 0
xotIfInIpMask:	255 . 255 . 255 . 0
xotIfOutIpAddr:	192 . 168 . 0 . 1
xotIfOutPort:	1998
xotIfMaxLinks:	5
xotIfMtu:	1456
xotIfAllowIncoming:	yes
xotIfSrcIpAddr:	. . .

Type: IP address (i.e. 199.155.32.17)

2. Define an XOI interface (x25OverIsdnIfTable)

- You have to specify one interface for every communication-direction
- For an incoming ISDN connection you have to set the parameter `xoiIfDirection` to in, the parameter `xoiIfSignalling` to 0x700 and the parameter `xoiIfL2Protocoll` to LAPB.
- For an outgoing ISDN connection you have to set the parameter `xoiIfRemoteNumber` to * and the parameter `xoiIfDirection` to out.
- For the values of the other parameters you can use the defaults.



Index	In...	Type	Direction	LocalNumber	RemoteNumber	StkMask	Signalling	L1Protocol	L2Protocol	PadP
0	27000	one_to_one	out	*		0xffffffff	0x0	auto	auto	0

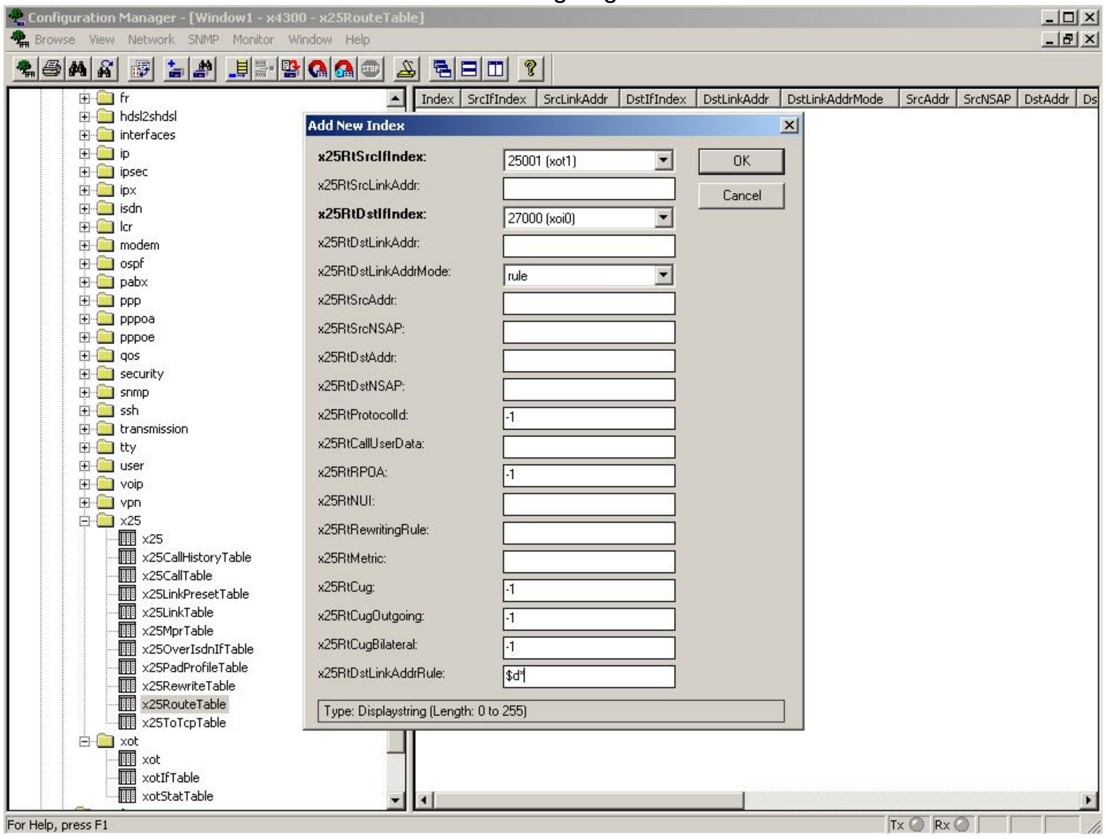
Add New Index

xoiIfIndex:	0 (REFUSE)
xoiIfType:	one_to_one
xoiIfDirection:	in
xoiIfLocalNumber:	
xoiIfRemoteNumber:	
xoiIfStkMask:	-1
xoiIfSignalling:	0x700
xoiIfL1Protocol:	auto
xoiIfL2Protocol:	lapb
xoiIfPadProfile:	
xoiIfModemProfile:	
xoiIfMaxCalls:	10
xoiIfL2Mode:	auto
xoiIfStatCommand:	nop
xoiIfMaxRetries:	0
xoiIfRetryTime:	0
xoiIfBlockTime:	0

After these configuration steps you should save the configuration and restart the configuration manager.

3. Define a route from an XOT interface to an XOI interface (x25RouteTable)

   - For every route you have to specify a separate entry. You can also enter a substitution rule for the ISDN number (this will replace the target ISDN number with the X.25 address of the X.25 packet)
   - Set the parameter `x25RtDstLinkAddrRule` to $d* (this will replace the ISDN number with the X25 address)
   - Set the parameter `X25RtDstLinkAddrMode` to rule (this will activate the substitution rule)
   - You have to set `x25RtSrcIfIndex` to the incoming XOT interface and `x25RtDstIfIndex` to an outgoing XOI interface


Configuration Manager - [Window1 - x4300 - x25RouteTable]

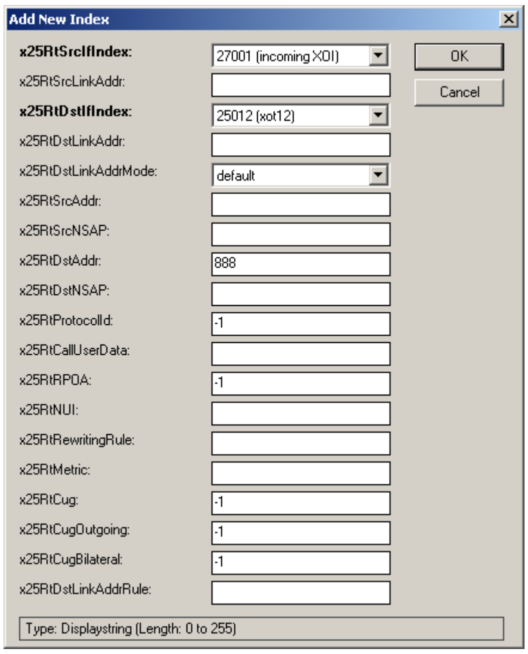
Add New Index

x25RtSrcIfIndex:	25001 (xot1)
x25RtSrcLinkAddr:	
x25RtDstIfIndex:	27000 (xoi0)
x25RtDstLinkAddr:	
x25RtDstLinkAddrMode:	rule
x25RtSrcAddr:	
x25RtSrcNSAP:	
x25RtDstAddr:	
x25RtDstNSAP:	
x25RtProtocolId:	-1
x25RtCallUserData:	
x25RtRPOA:	-1
x25RtNUI:	
x25RtRewritingRule:	
x25RtMetric:	
x25RtCug:	-1
x25RtCugOutgoing:	-1
x25RtCugBilateral:	-1
x25RtDstLinkAddrRule:	$d*

Type: Displaystring (Length: 0 to 255)

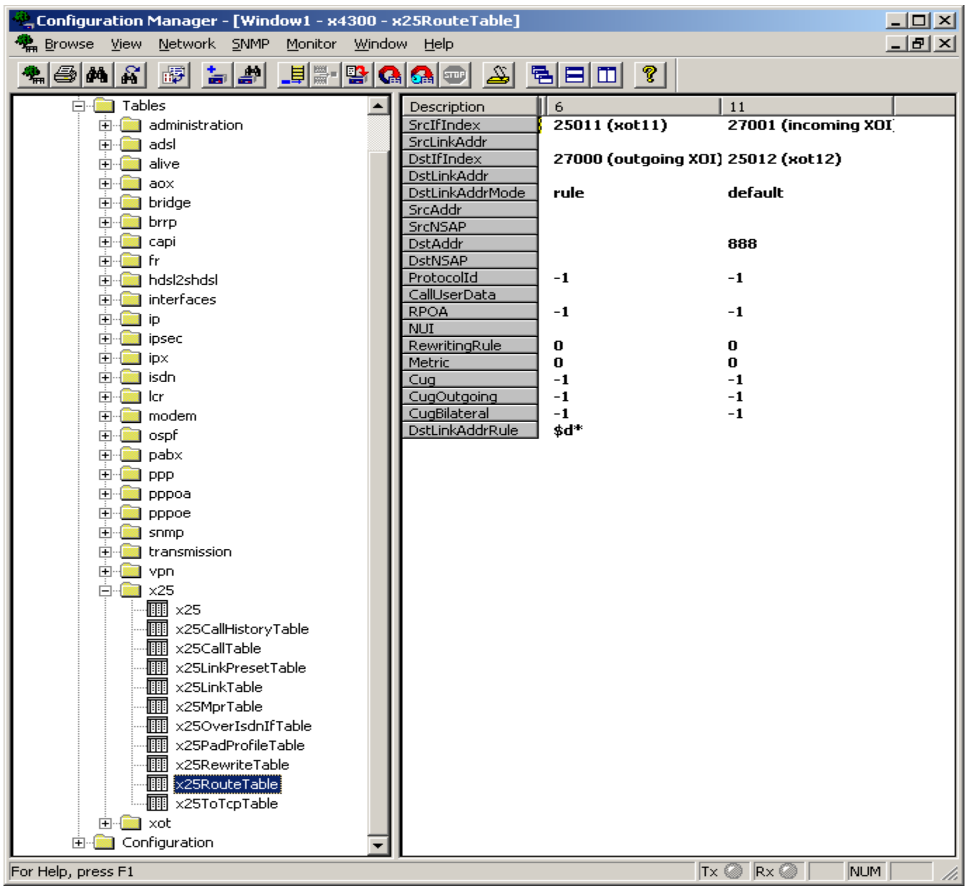
4. Define an route from an XOI interface to an XOT interface (x25RouteTable)

   - You have to specify for every route a separate entry.
   - For an incoming XOI connection you have to enter the X.25 address of the target station (parameter `x25RtDstAddr`). The BINTEC router uses this address for routing incoming X.25 packets to the right destination.
   - You have to set the parameter `x25RtDstLinkAddrMode` to default.
   - You have to set `x25RtSrcIfIndex` to the incoming XOI interface and `x25RtDstIfIndex` to an outgoing XOT interface.



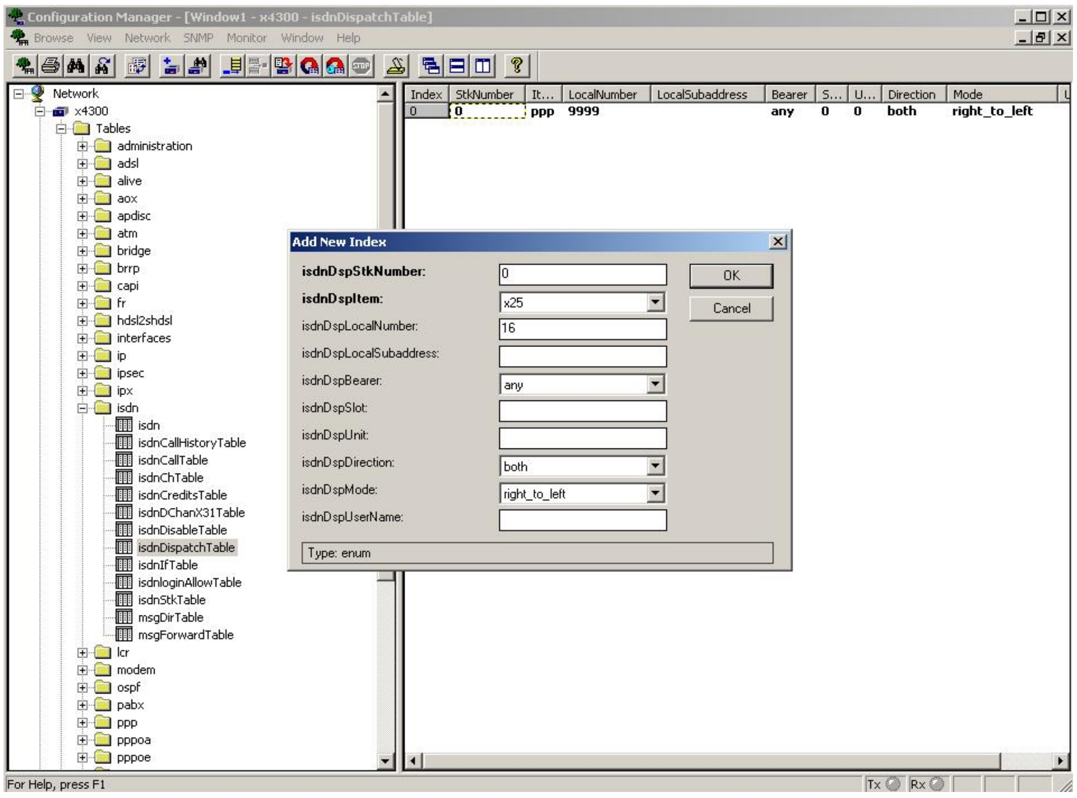
x25RtSrcIfIndex:	27001 (incoming XOI)
x25RtSrcLinkAddr:	
x25RtDstIfIndex:	25012 (xot12)
x25RtDstLinkAddr:	
x25RtDstLinkAddrMode:	default
x25RtSrcAddr:	
x25RtSrcNSAP:	
x25RtDstAddr:	888
x25RtDstNSAP:	
x25RtProtocolId:	-1
x25RtCallUserData:	
x25RtRPOA:	-1
x25RtNUI:	
x25RtRewritingRule:	
x25RtMetric:	
x25RtCug:	-1
x25RtCugOutgoing:	-1
x25RtCugBilateral:	-1
x25RtDstLinkAddrRule:	

Type: Displaystring (Length: 0 to 255)



Description	6	11
SrcIfIndex	25011 (xot11)	27001 (incoming XOI
SrcLinkAddr		
DstIfIndex	27000 (outgoing XOI)	25012 (xot12)
DstLinkAddr		
DstLinkAddrMode	rule	default
SrcAddr		
SrcNSAP		
DstAddr		888
DstNSAP		
ProtocolId	-1	-1
CallUserData		
RPOA	-1	-1
NUI		
RewritingRule	0	0
Metric	0	0
Cug	-1	-1
CugOutgoing	-1	-1
CugBilateral	-1	-1
DstLinkAddrRule	$d*	

5. Define the incoming call answering (IsdnDispatchTable)
   - Set the parameter `isdnDspStkNumber` to 0 and the parameter `isdnDspItem` to x25 (this will dispatch the call to the x25-interface).
   - Set the parameter `isdnDspLocalNumber` to the local number which handles the incoming call (MSN).



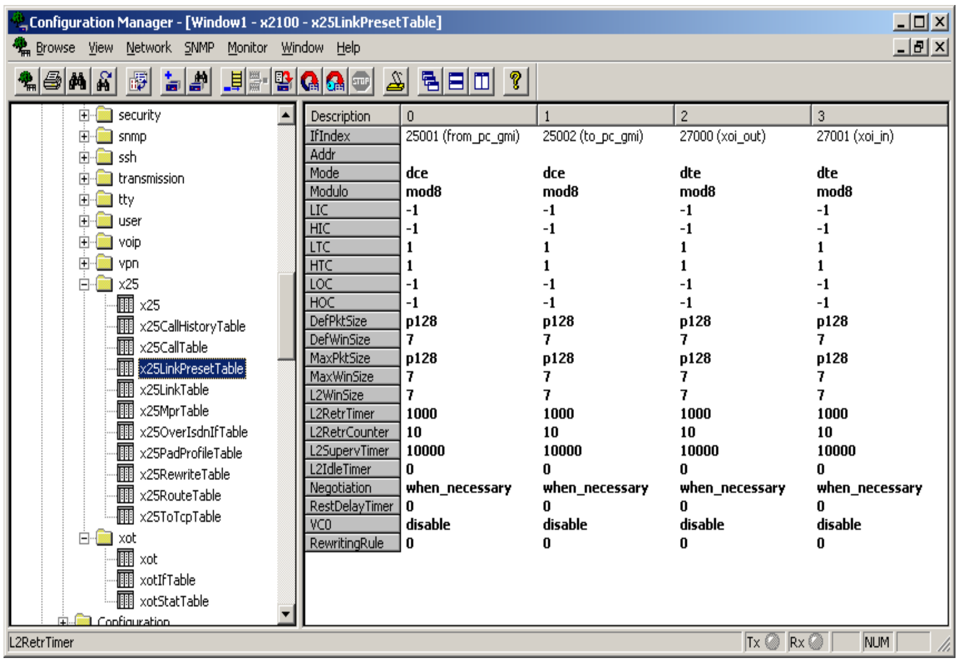
Index	StkNumber	It…	LocalNumber	LocalSubaddress	Bearer	S…	U…	Direction	Mode
0	0	ppp	9999		any	0	0	both	right_to_left

isdnDspStkNumber:	0
isdnDspItem:	x25
isdnDspLocalNumber:	16
isdnDspLocalSubaddress:	
isdnDspBearer:	any
isdnDspSlot:	
isdnDspUnit:	
isdnDspDirection:	both
isdnDspMode:	right_to_left
isdnDspUserName:	

Type: enum

7.3.2. Define for every interface default X.25 parameters (x25LinkPresetTable)

- For every XOI Interface you have to set the parameter `x25LkPrMode` to DTE
- For every XOT Interface you have to set the parameter `x25LkPrMode` to DCE
- Set the parameter `x25LkPrModulo` to mod8
- You can change packet size and windows size according national configurations



Description	0	1	2	3
IfIndex	25001 (from_pc_gmi)	25002 (to_pc_gmi)	27000 (xoi_out)	27001 (xoi_in)
Addr				
Mode	dce	dce	dte	dte
Modulo	mod8	mod8	mod8	mod8
LIC	-1	-1	-1	-1
HIC	-1	-1	-1	-1
LTC	1	1	1	1
HTC	1	1	1	1
LOC	-1	-1	-1	-1
HOC	-1	-1	-1	-1
DefPktSize	p128	p128	p128	p128
DefWinSize	7	7	7	7
MaxPktSize	p128	p128	p128	p128
MaxWinSize	7	7	7	7
L2WinSize	7	7	7	7
L2RetrTimer	1000	1000	1000	1000
L2RetrCounter	10	10	10	10
L2SupervTimer	10000	10000	10000	10000
L2IdleTimer	0	0	0	0
Negotiation	when_necessary	when_necessary	when_necessary	when_necessary
RestDelayTimer	0	0	0	0
VC0	disable	disable	disable	disable
RewritingRule	0	0	0	0

7.3.3. Example of BINTEC X4300 Configuration

Please see the text file `BINTEC_X4300.txt` (our current configuration) as an example. This textfile is to be found on the distribution CD, in the directory `_doc/ROUTER_CONF_EXAMPLES`.

8. rvs® Parameters

The function of the rvs® monitor can be influenced by the rvs® parameters. Their possible values and how to use them are described in this chapter. How to work with rvs® parameters is described in the User Manual.

The next section describes how to modify rvs® parameters on different platforms.

rvsX and rvs400: The values of the rvs® parameters should be set in the file `$RVSPATH/init/rdmini.dat`. These settings will be read every time the rvs® monitor starts and are valid over a session.

You can also use the Operator Console (rvscns) to set the parameters (valid only for a session).

rvsXP: Use rvsXP GUI to change the rvs® parameter. The following steps are necessary to reach the rvsXP parameter dialog: `rvsXP Administrator -> View -> Parameters`.

Alternativelly you can use in rvsXP the OperatorCommand dialog to change the parameters `rvsXP Edit -> OperatorCommand`). This possibility is equivalent to the OparatorConsole modus in rvsX and rvs400 is only over a session valid, too.

Following syntax is valid (for the file `rdmini.dat` and Operator Console):

Syntax:

```
setparm PARM=VALUE
```

or

```
sp PARM=VALUE
```

Example: `setparm ODTRACELVL=3`

No plausibility check is made on the value you specify and results are unpredictable if you specify invalid values or wrong data types.

Use

```
listparm name
```

or `lp name` to list one or more parameter values.

Example: `lp TCPIPRCV`

`name` can be

- the name of a parameter to list this one value,
- a pattern (see the next section in this chapter) to list all parameters whose names match this pattern or
- **ALL** to list all parameters (`list all`).

Some commands support patterns, which means that the values you specify for these parameters may include wildcards asterisk (*) to select more than one value at a time:

*	matches any number of arbitrary characters,

pattern must be enclosed in single or double quotation marks. For example, to list the execution priorities of all commands, enter

```
listparm "*prio"
```

and the Monitor will list the values of **BBPRIO**, **IEPRIO**, **IZPRIO**, etc. Specifying

8.1. rvs® Parameters′ Overview

The execution of the Monitor and its related components may be influenced by changing parameter values.

ACTPCOUNT **(only parameter for ActivePanel)**	the interval after which the statistical information about the active lines will be updated; the units are percentage of the actual filesize default: **10**
AECCHECK	check authority to execute (Monitor internal) command default: **0** (turned off)
AGENT_ HEARTBEAT	Only valid for rvs® SNMP Agent. Please refer to the rvs® SNMP Agent for more information. With this parameter enabled

	(0 = off; any value greater than 1 = on), rvs® monitor sends a heartbeat message to the TCP/IP address of the agent at regular intervals. When the rvs® monitor is idle, this parameter depends on the rvs® **SLEEP** parameter as follows: When the value of the **AGENT_HEARTBEAT** parameter is smaller than the value of the **SLEEP** parameter (default is 30 seconds), the value of the **SLEEP** parameter is adopted. There are no dependencies on the **SLEEP** parameter when rvs® Monitor is active; the actual value of the **AGENT_HEARTBEAT** is used. Default: **0** = off
AGENT_ LOGLEVEL	Only valid for rvs® SNMP Agent. Please refer to the rvs® SNMP Agent User Manual for more information. This parameter defines, which log messages are to be sent: 0 = none; 1 = all; 2 = error (message class: Error and Warning). Default: **0** (none)
AUTODECRYP	**AUTODECRYP = Y**: a received file is not checked for a header and not decompressed / decrypted automatically. The received file will be copied in directory \usrdat. You have to decompress / decrypt the files separate via COMSECURE. Values: **Y** (default), **N**
BBCREATE	creation of user notifications (`BB` command) default: **0** (turned off)
BBPRIO	priority of user notifications (`BB` command) default: **90**
BRICKOFTPTI	Time in seconds to wait for data by BRICK ISDN Adapter default: **20**

CALLINGNUMCHECK	**CALLINGNUMCHECK = Y**: additional to the Odette ID the call number / IP address of the caller (Calling Party) is checked. If the call number / IP address does not correspond to the expected value, the connection is terminated. Values: **Y**, **N** (default)
CDWAIT	time in seconds (0-5) before executing an OFTP **C**hange **D**irection after receiving a file default: **1** (1 second)
CHECKMAXP-SESSIONS	**CHECKMAXPSESSIONS = Y**: the number of simultaneous connections is restricted to the value defined in **PSESSION** station parameter. Values: **Y, N** (default)
CMDDELETE	remove each command and its related entries from database as soon as command ends or is being deleted default: **1** (turned on)
CNSMSGS	IDs of LOG messages to be sent to operator console. The following message codes are defined: **A** action **B** security **E** error **I** information **L** linedriver **O** ODETTE **R** report **S** severe error **W** warning **+** long messages default: **BEILORSW+**
CNTIE	CNTIE stands in connection with IECLEANTIME. After CNTIE processed commands the database will be checked for entries older than IECLEANTIME, that

	are not processed yet. Integer, Default **1000**
CODEIN	input code of a local file when creating a send entry default: **A** (for ASCII)
CODEOUT	output code of the file to be send. Should be the local code of the remote host; on UNIX/NT systems it is **A** ASCII; on OS/400 and OS/390 **E** EBCDIC. Default: **X** is LOCAL_CODE on the remote host.
COMPFLAGS	The version of offline compression. Possible values: `1` or 2. For files bigger than 4 GB is version 2 necessary. Normally you can use version `1 or 2`. Default: 2 **Note**: The parameters **COMPFLAGS** and **CRYPFLAGS** stand in correlation. If you use different values for **COMPFLAGS** and **CRYPFLAGS**, the smaller value will be taken.
COMTIMEOUT	only for rvs® Data Center: timeout for send jobs (rvscom). Default: **700** sec.
CRYPFLAGS	The version of encryption. Possible values: `1` or 2. For files bigger than 4 GB is version 2 necessary. Normally you can use version `1` or `2.` Default: 2 **Note**: The parameters **COMPFLAGS** and **CRYPFLAGS** stand in correlation. If you use different values for **COMPFLAGS** and **CRYPFLAGS**, the smaller value will be taken.
DBLOGMAXENTRIES	This parameter is only valid for rvs with an external database. rvs with an external database offers a feature to write the log messages into a database (parameter **LOGINDB** in the environment file `rvsenv.dat`, see User Manual). These log messages can be saved on the disc

	for not to charge the database table LT. This parameter sets the maximal number of log messages in the database. If this value is overlapped, the older part of log messages will be exported to the file `$RVSPATH/arcdir/dblogmessages.<timestamp>` and deleted in the database. Default: 50000 log messages
DBLOGMINENTRIES	This parameter is in correlation with the parameter DBLOGMAXENTRIES (see this table). If the log messages were exported, it is the parameter DBLOGMINENTRIES which decides how many massages will be kept. The latest messages will be kept. Default: 10000 (log messages)
DIALCOUNT	Only for ISDN/X.25. Number of dial attempts, when the line is busy. If these attempts are not successful, rvs will try to establish the connection with another line (network). See chapter about the alternative networks in the User Manual. This parameter is in correlation with the parameter **DIALCOUNT** (number of try attempts, see this table).
DIALRTIME	Time in seconds, before a new attempt on a “busy” X.25/ISDN line should take place. This parameter is in correlation with the parameter **DIALCOUNT** (see this table).
DISKSPACEREJECT	Number of Kilo Bytes, to be free in the directory `$RVSPATH/temp` for the receipt of files. See chapter 8.3 for more information. Default: 5000 (ca. 5 MByte)
DISKSPACEWARN	Number of Kilo Bytes, that has to be free in every rvs® directory, which will be checked. If the number of free Kilo Bytes is less than a value of this parameter the script `diskspacewarn` will be called. See chapter 8.3 for more information Default: 5000 (ca. 5 MByte)
DTCONNnn	wait periods until an unsuccessful

connection attempt is repeated.

nn is the number of unsuccessful attempts (**CNTRETRY** in `SK`). There need not be a parameter for all values of **nn**; if a particular one is not defined, the next smaller one that is found will be used.

format: MM/DD/YY HH:MM:SS

defaults: increasing time intervals, so that rvs® will not be kept busy trying to reach a station that may be having hardware problems. For longer wait periods, minutes have been added to the defaults, so that retries will not occur at precisely the same time:

DTCONN01	"00/00/00 00:01:00"
DTCONN02	"00/00/00 00:02:00"
DTCONN03	"00/00/00 00:03:00"
DTCONN05	"00/00/00 00:05:00"
DTCONN07	"00/00/00 00:07:00"
DTCONN10	"00/00/00 00:10:00"
DTCONN15	"00/00/00 00:15:00"
DTCONN20	"00/00/00 00:20:00"

In addition, **DTCONN01** is the wait period for all other rvs® commands.

DTCOPY

Time interval: after the expiration of interval, the internal feed starts again, if there is an error with copying received files from \temp directory to \usrdat directory .

Format: YY/MM/DD hh:mm:ss, default: **00/00/00 00:10:00**, i.e. 10 minutes

EERP_IN

Wait for a receipt from the partner

NEVER partner does not send EERP, so a send request ends with the correct transmission without waiting for the receipt

NORMAL wait for receipt, end send request when receiving receipt

Default: **NORMAL**

EERP_OUT

Handling for sending a receipt

NEVER partner does not expect EERP, so don't create a receipt.

IMMEDIATE create a receipt and start a session, if no session is available.

NORMAL create a receipt, but wait for a session to transmit (suggested)

SYNC force transmission of a receipt (EERP) for a received file in the same session in which the file was received.The connection is not closed unless the EERP is created (after successful file delivery). The rvs parameter `SYNCDL` holds a time value in milliseconds that shall be waited before it is checked whether the EERP is ready to be sent. The number of wait periods is set by the rvs parameter `SYNCTO`. The rvs parameters `SYNCTO` and `SYNCDL` should be configurated in `$RVSPATH/init/rdmini.dat`.

Example: If `SYNCDL=400` and `SYNCTO=5`, a time span not exceeding 5 times 400ms is waited until the connection is closed. If in this time the EERP is created, it is transmitted, and the connection is closed immediately.

HOLD create a receipt, but do not send it. When a receipt is released, send it in the next session.

HOLD_IMMED create a receipt, do not send it. When a receipt is released, create a session and send it immediately.

Please read User Manual for more information about releasing the receipts in the status **HOLD** or **HOLD_IMMED**.

Default: **NORMAL**

Note: Default for station table (OP parameters) is **IMMEDIATE** (see User Manual, Station configuration, Odette parameters).

EFIDGAPTIMEOUT This parameter defines the interval at which rvs starts the transmission again, if the status of the transmission is unclear.

	Format: YY/MM/DD hh:mm:ss, default: **00/00/00 01:00:00**, i.e. 1 hour
FORCEDEND	Halting of the Monitor with Monitor-Stop: Immediate cancellation, even if the transmitter and receiver are active. **NOTE**: If the parameter is set at "1" the Monitor will immediately stop. default: **0**
HEAVYDUTY	This parameter defines the interval at which the rvs system writes an entry in the log file, if the system is overload. I.e. the rvs sytem needs more time for the duty as forced with **HEAVYDUTY**. Format: YY/MM/DD hh:mm:ss, default: **00/00/00 01:00:00**, i.e. 1 hour Only with ISAM systems
IECLEANTIME	IE's (information entries) from aborted or unprocessed commands, that are hold in database longer than defined in **IECLEANTIME** time interval, will be deleted. Related to **CNTIE** Time interval: Format: YY/MM/DD hh:mm:ss, default: **00/00/14 00:00:00**, i.e. 14 days
IEPRIO	priority of IE-commands. Possible values: 1 – 100. default: **50**
INITCMDS	execute initialization commands default: **1** (turned on)
IZPRIO	priority of IZ commands default: **40**
JSERRHOLD	rvs analyses this parameter when a send entry fails and rvs launches a job start after send attempt. The value of this parameter decides whether the send entry should be hold or not. If the parameter has the value **Yes** rvs holds the send entry (sets the status to „hold“). If the parameter has the value **No** rvs

	tries to finish the send entry.
	Standard: **No**
KEEPDAYS	number of days, after which deleted and ended commands and their related information may be discarded during database cleanup
	default: **7**
LDSNPRIO	send priority for big files. Possible values: 1 -100. If the value is smaller, the priority is higher. This parameter should be used with parameters **SDSNPRIO** and **SDSNMAX**. If **LDSNPRIO** is higher than **SDSNPRIO** (default), the smaller files have priority.
	default: **50**
LID	local station ID
	default: supplied during database initialization
LITRACELVL	request line tracing (between OFTP and network): 0 no tracing 1 minimum tracing (line driver events etc.) for station specified in parameter **SIDTRACE** 2 detailed tracing (incl. hex dump of data) for station specified in parameter **SIDTRACE** 3 detailed tracing for all stations.
	default: **0**
LMPRIO	priority for LOG messages as external LM commands
	default: **20**
MASKRCV	Only for internal use
MASKSND	Only for internal use
MAXCMD	max external commands read once
	default: **10**
MAXRECL	maximum record length for data sets with record format **F** or **V** to be received
	default: **99999**

MAXSENDERS	maximum number of concurrent senders. If **MAXSENDERS**=**0**, no sender will start. See also the parameter **PSESSIONS** in the NK station table, chapter (if the value of PSESSIONS -1 is, the value of the global parameter MAXSENDERS should be used) default: **1**
MAXX25RCV	maximum number of concurrently active or prestarted "listening" receiver processes for X.25 native communication default: **0**
MONTIMEOUT	This parameter is only for rvs Data Center possible. It is a Timeout for rvs monitors. Default: 700 seconds
MSGPRIO	send priority for operator to operator messages default: **60**
MWSTART	If wanted rvs® and rvs® Middleware can be started together. Possible values: 1 (rvs® and rvs® Middleware start together) 0 (no common start) default: **0**
MWSTOP	If wanted rvs® and rvs® Middleware can be stopped together. Possible values: 1 (rvs® and rvs® Middleware will be stopped) 0 (no common stop) default: **0**
MWTIMEOUT	rvs® Middleware should registrate own activities in the RI table of rvs® database. It is a timeout for this registration. If no registration takes place for the time, which is indicated in the default value of this parameter, rvs® Middleware will be stopped.

	default: **700 seconds**
NUMREDOLOGS	number of generations of `redo.log` files Possible values : 1 - no limit. default: **NOLIMIT**
NUMRLOGS	number of generations of `rlog.log` files default: **NOLIMIT**
NUMRLSTATS	number of generations of `rlstat.log` files default: **0**, no limit
OCREVAL	ODETTE credit value = window size of OFTP: Maximum number of sent blocks without confirmation default: **99**
ODTRACELVL	request line tracing (between sender and OFTP): 0 no tracing 1 minimum tracing (request names, only) for station specified in **SIDTRACE**. 2 detailed tracing (parameter values etc.) for station specified in **SIDTRACE**. 3 detailed tracing but for all stations. default: **0**
OEXBUF	maximum ODETTE exchange buffer size in bytes (the largest ODETTE cmd (**SFID)**); this parameter can be configurated by customizing ODETTE Parameter in the OP table, too (see chapter about ODETTE parameters in the User Manual). default: **2000**
OKPRIO	priority for operator commands. Possible values: 1 – 100. default: **10**
ORETRY	indicates the ODETTE error group for which a retry will be initiated after a request has been interrupted. It is a bit

	field from left to right: 1 – retry will be initiated; 0 – retry won’t be initiated. default: **10111000111011111111** The bits stand for: • 1 – transmission is interrupted • 2 – file not found or can’t be opened • 3 – file can not be read • 4 – „File size is too big″ in SFNA with retry is allowed • 5 – „Unspecified reason″ in SFNA with retry is allowed • 6 – „File size is too big″ in SFNA with retry is not allowed • 7 – „Unspecified reason″ in SFNA with retry is not allowed • 8 – „File size is too big″ in EFNA • 9 – „Invalid record count″ in EFNA • 10 – „Invalid byte count″ in EFNA • 11 – „Acess method failure″ in EFNA • 12 – „Unspecified reason″ in EFNA
OTIMEOUT	ODETTE time-out value (in seconds). Possible values: 1 – 100. default: **600**
QEPRIO	priority of `QE` commands. Possible values: 1 – 100. default: **30**
QSPRIO	priority of `QS` commands should lie between **MSGPRIO** and **SDSNPRIO** default: **30**
RECVBLOCKS	number of buffers or records that the Receiver writes before closing temporary data set. default: **1000**
REDOMAXSIZE	maximum file size in bytes for the file `$RVSPATH/arcdir/redo.log`

	default: **1000000**
RLCOMAXSIZE	maximum file size for consol messages `rlco.log` default: **1000000**
RLDBMAXSIZE	maximum filesize for logging of database actions `rldb.log` default: **1000000**
RLOGMAXSIZE	maximum filesize for log messages `rlog.log` default: **1000000**
ROUTING	Sometimes it is advantageous not to allow OFTP routing. This is possible by setting the rvs parameter `ROUTING` for single stations in the OP table. Using the same parameter in the file `$RVSPATH/init/rdmini.dat`, you can suppress/allow it or for all partner stations. **I** (means IN): the incoming file transmission from the partner (e.g. `XXX)` to the remote partner (e.g. `REM1)` via our local station (e.g. `LOC)` is permitted (`XXX ↦ LOC ↦ REM1`); not permitted is the outcoming routing e.g. for the partner `REM2` via `REM1` (`LOC ↦ REM1 ↦ REM2`). **O** (means OUT): partner stations can't use your local station as router. Permitted is the outgoing routing e.g. for the partner `REM2` via `REM1` (`LOC ↦ REM1 ↦ REM2`). Not permitted is: the incoming file transmission from the partner e.g. `XXX` to the remote partner e.g. `REM1` via our local station e.g. `LOC` (`XXX ↦ LOC ↦ REM1`). **B** (means BOTH; IN and OUT): normal OFTP routing **N** (means NEVER): routing in both direction IN and OUT forbidden. default: **B**
RSTATMAXSIZE	maximum filesize for statistical logs `rlstat.log`

	default: **0**, no limit
RVSDIAEXTENDED-MODE	With this parameter you can get additional details of routed receive jobs in `rvsdia`. Possible values: **1**, **Y**, **y**, **J**, **j** for additional details **0** basic display (default)
SCPRIO	The frequency of the checking the ServiceProvider output by the rvsX monitor (in the directory `SPOUTDIR` (e.g. `/home/rvs/temp/out`), see the file `$RVSPATH/rvsenv.dat` for the value of `SPOUTDIR`. default: **10**
SDSNMAX	maximum size for a file to be considered short (in units of 1024 bytes) default: **100**
SDSNPRIO	send priority for small files. default: **40**
SECURITY	This parameter specifies the usage of the data encryption. It can be set for the corresponding station in the `OP` table or as rvsX global parameter in `RVSPATH/init/rdmini.dat` (then it is valid for stations without own `SECURITY` entry). **NO** Encryption isn't possible. If a send job requests encryption the job is cancelled accompanied by an error message. **OPT** Encryption is optional and may be set by each send job **FORCED** Encryption is enforced. If a send job does not switch on encryption, a warning message is created and encryption is switched on. If the partner station sends an unencrypted file the reception is denied. A send job for a partner station is handled corresponding to the `SECURITY` entry for this station. It is not important whether a partner station is a neighbour station or is reached via routing.

	More about encryption please read in User Manual. default: **OPT**
SENDBLOCKS	number of buffers or records that the Sender sends before looking at **FORCEDEND** again. default: **1000**
SEPRIO	priority for new `SE`s should be at least as high as the highest priority of what can be trans mitted by a `SE` default: **50**
SIDTRACE	ID of station that shall be traced (if **LITRACELVL** or **ODTRACELVL** are set min. to **1** or **2**. default is " " (3 blanks). If you need to trace incoming data, **SIDTRACE** must be set equal to the local station ID, **LID**.
SLEEP	Monitor suspension time in seconds when there is no work to do, the Monitor waits this period of time, before checking, whether a command waits processing. default: **30**
SSCREATE	creation of a send statistics record for each transfer attempt default: **0** (turned off)
STATCHECKINT	Time interval at which an admission of the system status takes place Format: YY/MM/DD hh:mm:ss, default: **00/00/00 01:00:00**, i.e. 1 hour
STATISTICS	creation of a send statistics record in the statistics log file (`rlstat.log` for rvsNT and rvsX) 0 no statistics log file 1 short form 2 detailed form of statistics 3 short form of statistics inclusive routed transfers 4 detailed form of statistics inclusive routed transfers

	5 new parameters such as fileformat, state of transmission, numer of dial tries
	6 statistics about deleted entries (by the user), too
	7 includes all of '6' inclusive routed transfers
	default: **2** (detailed statistics turned on)
SYNCDL	Please read about parameter **EERP_OUT** in this chapter.
	default: **500**
SYNCTO	Please read about the parameter **EERP_OUT** in this chapter.
	default: **120**
TCPIPRCV	maximum number of (concurrently) prestarted "listening" processes for TCP/IP communication:
	0 no TCP/IP receiver will be started
	n TCP/IP receiver will be started
	default: **1**
TIMESTAMP	creation of a timestamp to destinguish data sets with the same dsname
	0 000-999 (counter for MS DOS file names)
	1 000000-999999 (counter)
	2 Thhmmss (Time)
	3 Dyymmdd.Thhmmss (Date and Time)
	4 Thhmmssmsms (Date and Time in milliseconds
	default: **2** (only time)
TMAXCON	TCP/IP maximum number of connections
	0 (no limit)
	default: **0** (no limit)
TRACEALWAYS-TOFILE	With this parameter you can store all line traces in a file. The severe line errors will always be traced into a file (also without the configuration of this parameter).

	The trace files will be stored in the directory `$RVSPATH/tracedir` under the name `traceXXX.log` (where XXX is the number of the rvscom process ID; this ID is also to find in the rvs MonitorLog file `$RVSPATH/system/rlog.log`). Possible values: **0** (off); **1** (on). Default: **0**. **Note**: Please refer to the chapter “Memory Line Traces” in the Reference Manual for more information on TRACE parameters.
TRACECONERR	Each connection error will be written into a trace file `$RVSPATH/tracedir/traceXXX.log` Possible values: **0** (off); **1** (on). Default: **0**.
TRACEMAXITEM	Number of records in memory. See the chapter 2.13 for more information. Default: 1024 (records).
TRACEMAXSIZE	Maximum size of traces in the memory (unit: kB). See the chapter 2.13 for more information. Default: 1024 kB
TRACEOFF	Traces in memory will be written or not. Possible values: **0**: traces will be written; **1**: traces will not be written. Default: **0**
TSTODPRCT	percentage of non-error returns from ODETTE simulation program when rvs® runs in testmode; -1 requests prompting for return values. default: **90**
VDSNCHAR	range of allowable charactersto be transferred within an ODETTE transmission: • **ALL**: no restrictions • **OFTPUNIXS**: all capital letters, digits and the special characters . - • **UNIX**: all letters, digits and the special characters # _ - + . • **ODETTE**: all capital letters, digits and

	the special characters () - . / & • **CHECK_RE**: same as ALL, but it is necessary that a RE exists
	default: **ODETTE**
VFTYP	the way how data sets (with a fixed or variable format) will be converted.
	V rvs® internal format, only useful for rvs® for variable and fixed formats
	S format of ft-SINIX, useful also for ft-SINIX
	T text format, each line is terminated by Nl (line feed); each line is converted into one output record. The record length is defined in **MAXRECL**. Please see Reference manual, chapter "How to work with rvs Batch Interface", section "Command `SEND`" for more information how to use the parameters **MAXRECL** and **VFTYP**.
	D files in fixed format are transmitted as binary files and files in format variable are transmitted as text files.
	default: **V** (rvs® internal format)
XMCREATE	creation of LOG messages with detailed information about what was transferred from and to whom after each successfull data set send or receive
	default: **1** (turned on)

8.2. Parameter STATISTICS

The **STATISTICS** parameter controls the creation of the `$RVSPATH/db/rlstat.log`, statistics log file, which contains entries regarding sent and received files. You can use this file for archiving purposes.

Like any other global rvs® parameters, the **STATISTICS** parameter is to be defined either in the `$RVSPATH/init/rdmini.dat` file or edited via the rvs® console for the current session only (see chapter 8).

Values can range from 0 to 7.

STATISTICS=0 prevents the creation of this file.

STATISTICS=1 signifies the `rlstat.log` file contains a line per received or sent file each with file name, date, time and sender/receiver ID.

Example:
```
RCV XPSKK 2005/02/14 08:59:14
/home/skk/rvs/global/usrdat/RVSENVALT.DAT
```

Value	Meaning
RCV	Direction: RCV (receive) Also possible: SNT (send)
XPSKK	RCV direction: Receiver station ID; SNT direction: sender's station ID
2005/02/14 08:59:14	Date and time: Receiving: Delivery of the file into the rvs® usrdat directory, sending: Provision of send order in the rvs® database.
/home/…/RVSENVALT.DAT	Path (local)

STATISTICS=2 creates the same file, but with additional extended information such as the file name for the transmission (virtual file name), the file size and command numbers for SE, SK (send) or IE, IZ (receive).

Example:

```
RCV XPSKK 31857 31860 2005/02/25 14:47:51
RDSTAT.DAT /home/skk/rvs/global/usrdat/RDSTAT.DAT
666
```

Value	Meaning
RCV	Direction: RCV (receive) Also possible: SNT (send)
XPSKK	RCV direction: Receiver station ID; SND direction: sender's station ID
`31211 31215`	Command number (IE and IZ for reception, SE and SK for transmission).
`2005/02/14 08:59:14`	Date and time: Receiving: Delivery of the file into the rvs® `usrdat` directory, sending: Provision of send order in the rvs® database.
`RVSENVALT.DAT`	VDSN (virtual file name used for transmission).
`/home/…/RVSENVALT.DAT`	Path (local)
`666`	RCV direction: Number of received bytes; SNT direction: Number of sent bytes

STATISTICS=3 generates the `rlstat.log` file with the same entries as **STATISTICS=1**.

STATISTICS=**4** is the same as **STATISTICS**=**2**, but routed file transfer (source and destination zone) will be logged too.

Example:

```
SNT MB01 MBX-MB01 31857 31860 2005/02/25 14:47:51
RTEST /home/skk/rvs/out/RTEST.DAT 666
```

In the above example, station `MBX` sends the `RTEST.DAT` file to station `MB01` (`MBX-MB01`).

STATISTICS=**5** generates a more detailed output in `$RVSPATH/db/rlstat.log` with new entries such as file format, state of transmission and number of dial attempts.

Example:

```
RCV XPSKK 0000031918000003l921 - RDSTAT.DAT XPSKK
2005/02/25 14:52:04  2005/02/25 14:52:04
/home/skk/rvs/global/usrdat/RDSTAT.DAT 0 2208 U 0
- - R
```

Value	Meaning
RCV	Direction, possible values: • RCV (receive), • SNT (send).
XPSKK	RCV direction: Receiver statio SNT direction: Sender station ID.
00000319180000031921	10-digit command number (IE and reception, SE and SK for transmissic
2005/02/14 08:59:24	Date and time: Receiving: Delivery file into the rvs® usrdat dir sending: Provision of send order in th database.
2005/02/14 08:59:34	Date and time of EERP (Er End-Response) According ODETTE a file has been comp received only when the EERF been received.
RVSENVALT.DAT	VDSN (virtual file name use transmission).
/home/…/RVSENVALT.DAT	Path (local)
2208	Number of sent bytes
2208	File size (in bytes)
U	File format: T = text file; U = unstru (binary) file; V = variable record lengt fixed record length.
0	Number of attempts to send
-	Status; also possible: en for er
-	The reason for deletion (commen text), when delcmd was specified.
R	Direction: RCV (receive) SNT (send)

STATISTICS=**6** produces a more detailed output about deleted entries (by the user) with the reason of deletion (if specified with `delcmd`).

STATISTICS=**7** is identical to **STATISTICS=6**, including routing

8.3. The robustness of rvs® concerning disk space

rvs® from the version 4.05 enables the possibility of disk space checking of all important rvs® directories. This increases the robustness of rvs® concerning disk space problems.

This functionality allows the operator to react early enough to lack of free disk space. In case of low disk space you can see in the Monitor Log a message, that informs you in which rvs® directories is not enough disk space. Depending on the degree of disk space lackness you have the possibility to dispay a warning or an error message. After an error rvs® will be stopped.

Example of a warning message:

```
W:  <DISKSPACEWARN   >Diskspace an /home/uitt/rvs/db, /home/uitt/rvs/temp
, /home/uitt/rvs/tracedir , /home/uitt/rvs/usrdat/ ,
/home/uitt/rvs/temp/in ,
 /home/uitt/rvs/temp/out , /home/uitt/rvs/temp/temp low, needed  11021504
kb , available 8610144 kb.
```

Example of an error message:

```
W:  <DISKSPACEERR    >Diskspace an /home/uitt/rvs/db ,
/home/uitt/rvs/temp , /home/uitt/rvs/tracedir , /home/uitt/rvs/usrdat/ ,
/home/uitt/rvs/temp/in , /home/uitt/rvs/temp/out ,
/home/uitt/rvs/temp/temp extrem low, needed  11021504 kb , availiable
8609792 kb.
```

After an error rvs® will be stopped and the following message will be dispayed:

```
A:  <OK_READ         >  [rvsstop] stop rvs=force
I:  <OK_CMD_DONE     >  [rvsstop]  'stop' done.
I:  <XMT_STOP_NORMAL >  Master Transmitter stops normal.
I:  <RVS_TERMINATION >  RVS Monitor terminates. (return code: 4).
```

In the case of a warning the script diskspacewarn will be called. In the case of an error another script with the name `diskspaceerr` will be called. The both scripts are examples from the directory `$RVSPATH/system` and can be adjusted to your needs. These scripts must be stored in the directory `$RVSPATH/system` and also named exactly as mentioned above.

Note: Scripts for UNIX platforms end with `.sh` (shell files) and scripts for windows end with .bat.

Example (`diskspaceerr.sh`):

```
echo "1 >$1<  2 >$2< 3 >$3<" >> $HOME/diskerr.log
echo "DISKSPACEERR : low disk space on >$1< ( needed $2 kb,
available $3 kb)" >> $HOME/diskerr.log
```

```
rvsstop -f
```

In this example script the log message will be additionally logged in a file `diskerr.log` and immediately afterwards rvs® will be stopped.

How to define a warning (how much of free disk space in rvs® directories should be available) can be configured by the parameter `DISKSPACEWARN` (see the table at the end of this chapter). An error is to be defined by the parameter `DISKSPACERR`.

The following rvs® directories will be regulary checked for free disk space: `DB`, `TEMP`, `TRACEDIR`, `USRDAT`, `SPINDIR`, `SPOUTDIR` und `SPFILEDIR`. The time interval for the checking should be configurated by the parameter `DISKSPACEDELAY` (see the table at the end of this chapter).

If the free disk space in the directory `$RVSPATH/temp` is low for the receipt of a file, the message “file too big” will appear in the Monitor Log. How much of free disk space has to be available for the successful receipt is to be configured with the parameter `DISKSPACEEJECT`.

Example (Monitor Log with the message “file too big” on the sender side):

```
A: 2006/03/15 14:15:19  <NEW_CMD_CREATED >  SK(125) created by SE(124).
A: 2006/03/15 14:16:26  <SENDER_STARTED  >  SK(125) Sender started to
SID(WOB_AIX) (using Prot(TCP/IP)).
A: 2006/03/15 14:16:26  <CALL_OUT        >  1196: outgoing call to
SID=WOB_AIX ...
O: 2006/03/15 14:16:27  <CONNECT         >  Sender: Connection with
Station 'WOB_AIX' with Credit=99, Odette Buffer=2000, OFTP compression
established.
O: 2006/03/15 14:16:27  <OFTP_SEND       >  Send:SK 125 'TESTTEST(060315
141519)' from:'LOC' Destination:'WOB_AIX' FORMAT=U RESTART=0.
A: 2006/03/15 14:16:27  <RPS_TERMINATION >  SK(125) Sending ended code 4
   (transmission rejected by neighbor (SFNA or EFNA))
O: 2006/03/15 14:16:28  <DISCONNECT      >  Connection as (send) to
Station 'WOB_AIX' ended.
I: 2006/03/15 14:16:51  <SERR_ODETTE     >  rpm: Node WOB_AIX rejected
Odette transfer for SK(125).
       Send File Negative Answer - retry allowed : file size is too big
```

Example (Monitor Log with the message “file too big” on the receiver side):

```
I:  <INCOMING_CALL   >  Incoming call received: DIEXP
O:  <CONNECT_IND     >  Responder: Connection with Station 'DIEXP' with
Credit=9
9, Odette Buffer=2000, OFTP compression established.
E:  <SHOW_CAUSE      >Error occured when starting or ending file transfer
(CAUSE
=6):  SFNA: file too big
O:  <DISCONNECT      >  Connection as (receive) to Station 'DIEXP' ended.
```

Here is the overview of the new rvs® parameters concerning the robustness of disk space:

Parameter	
DISKSPACEERR	Number of Kilo Bytes, that has to be free in every rvs® directory, which has to be checked. If the number of free Kilo Bytes is less than a value of this parameter the script `diskspaceerr` will be called. Default: 1000 (ca. 1 MByte)
DISKSPACEWARN	Number of Kilo Bytes, that has to be free in every rvs® directory, which has to be checked. If the number of free Kilo Bytes is less than a value of this parameter the script `diskspacewarn` will be called. Default: 5000 (ca. 5 MByte)
DISKSPACEDELAY	Time interval in seconds for the apperance of `DISKSPACEWARN` in the Monitor Log and the call of the script `DISKSPACEWARN`. Default: 600 seconds (10 minutes)
DISKSPACEREJECT	Number of Kilo Bytes, to be free in the directory `$RVSPATH/temp` for the receipt of files. Default: 5000 (ca. 5 MByte)

8.4. rvs® Parameter Values

rvs® contains a number of optional and security related features which you may not need (all the time) at your installation. When activated, these features consume computer resources (processor time and disk accesses) and thus may dramatically influence performance of rvs® components.

As an example, consider transmission of a large data set. To be able to resume transmission after a line failure without having to start at the beginning of the file all over again, the Receiver periodically closes the incoming data set, and both Sender and Receiver store the number of transmitted bytes or records in the database. The frequency of these actions is determined by parameters **SENDBLOCKS** and **RECVBLOCKS**.

Reopening and positioning a large data set involves quite a number of disk access operations and therefore is very time consuming[1]. So, if most of your communication lines are very stable, you will want to set these parameters to very large values effectively disabling the restart feature of rvs®.

If, on the other hands, most of your lines tend to break down every few minutes, you will want to make sure that whatever has been transmitted once, will not have to be transmitted again. Note, that a large value of **SENDBLOCKS** may also increase the time before Senders terminate after Monitor has been stopped.

Defaults have been chosen, so that rvs® will work securely and with most options enabled.

8.4.1. Safety, Resource Consumption and Performance

Besides **RECVBLOCKS** and **SENDBLOCKS** which have been discussed above, there are several other parameters that influence the balance between safety, resource consumption, and performance.

OCREVAL (recommended window size 99) and **OEXBUF** (recommended size 2048 bytes) influence the overhead incurred by the ODETTE protocol; the larger these values, the less overhead_but the more memory will be required for Sender and Receiver. These values may be negotiated down at the start of each transmission, so that unilateral changes may have no effect. What you really determine is the maximum amount of memory you are willing to allocate to ODETTE.

Searching in a large database generally takes longer than looking for something in a small one; a larger database, however, retains more information on completed transmissions. **KEEPDAYS** determines the number of days you want to keep information about ended or deleted transmissions (unless you use cleanup days=n, explicitly specifying the retention period in the command itself).

For **CMDDELETE**=1, all related entries will be removed physically from the database when a commands ends or when it is (logically) deleted. This keeps the size of the database as small as possible. If you choose this option, you should leave **XMCREATE** at its

[1] Transmission time for a 4.5 MB data set between two OS/2 nodes was reduced by about a factor of 10 (from more than an hour to a few minutes) by changing the values of these parameters from `10' to `10000'.

default value (`1'), so that detailed LOG messages will be created after sending or receiving a data set and all users should have access to the log data set (`$RVSPATH/db/rlog.log`) to be able to look at these messages, because the dialog interface will be unable to display any information about completed transfers. Consider using this option for continuous unattended operations.

The Monitor's reaction time to new events is determined by **SLEEP**; this may influence for example, how long it takes, before the Monitor starts acting on an operator command. **SLEEP** is the period of time (in seconds) that the Monitor is suspended when there is nothing to do for it and the longer you choose this period, the less it will interfere with your other applications, but the longer you may have to wait, before it starts processing your requests. The shorter you choose this period the higher is the unproductive overhead produced by scanning the database when there is nothing to do.

The time until the Monitor restarts an unsuccessful or aborted transmission is determined by the **DTCONNxx** parameters. The smaller these values, the sooner the transmission will start after the line is up again but the more computer time may have been wasted on unsuccessful attempts until the line is restored.

8.4.2. Limit Number of Concurrent Senders

If your system is very busy or when you know, that one or more of your neighbors cannot accept more than a few incoming calls at the same time, then you want to limit the number of Senders that rvs® is allowed to execute at the same time.

MAXSENDERS tells MasterTransmitter `rvsxmt` how many Senders may run concurrently. When this number has been reached, it waits until a Sender terminates before starting the next one. If **MAXSENDERS** is set to **0**, no Sender will be started at all. This is useful if only the partner station should establish the connection and get the queued data sets. Use the operator command `activate`, to send data to a specific station even if **MAXSENDERS** is set to **0**.

8.4.3. Limit Number of Concurrent X.25 or ISDN Receivers

You must specify the number of concurrently active X.25 and ISDN receivers. A small number is adequate for low traffic, a higher number is required if you must be able to receive data on several connections in parallel. However, there cannot be more X.25 receivers, than virtual channels are available on your X.25

multichannel or, in case of ISDN, there cannot be more receivers than B-channels are available. Because Senders also occupy virtual channels or B channels in ISDN repectively, the number of concurrent receivers should be limited to half the total number of channels.

MAXX25RCV tells MasterTransmitter rvsxmt how many Receivers must run concurrently. It prestarts as many X.25 (ISDN) receivers as indicated. If a receiver terminates, MasterTransmitter will start a new receiver, which in turn will wait for incoming calls. **MAXX25RCV**, if set to zero, prevents any incoming X.25 or ISDN traffic. If only SNA-LU6.2 or TCP/IP is used, it must be set to zero.

On product systems, if **MAXX25RCV** is greater than 1, you have to define additional entries in the X.25 routing table.

8.4.4. TCP/IP Receiver

If you want to communicate via TCP/IP, rvs® has to start a TCP/IP receiver, wich waits on incoming calls. You must set the value of the parameter **TCPIPRCV**. If only LU6.2 or X.25/ISDN is used, it must be set to zero. If a TCP/IP receiver accepts an incoming call, MasterTransmitter will start a new receiver on the same port, which in turn will wait for incoming calls. So, on each port, you can accept as many calls as indicated by the values MAX_IN for your local station in your stationtable.

8.4.5. Optional Features

Providing these optional services takes time and uses up disk space; so, you may wish to turn them off, if you do not need them.

AECCHECK is a flag which tells the Monitor, to check whether the originator of the command currently being processed has the authority to issue this particular command. In a (future) multi-console environment, this could be used to prevent certain consoles from stopping the Monitor, for example. Currently, this feature is not fully supported, so **AECCHECK** should remain at **0** (turned off).

Statistics records will be created for every attempted transfer when flag **SSCREATE** is turned on (**SSCREATE**=**1**). These records contain the station ID of the neighboring rvs® node as well as time and completion code of the attempted (or completed) transfer. **SSCREATE**=**0** prevents generation of these records. Currently, no utility to analyze these records is provided.

XMCREATE (create xfer message) controls generation of detailed information about successful transfers in the system log (`$RVSPATH/db/rlog.log`). If **XMCREATE**=**1** (the default), a log message will be written, whenever a data set is successfully sent to a neighboring node (even before a receipt has been received), whenever a send entry completes (after receiving receipts from all recipients), and whenever a data set has been delivered to a local user. **XMCREATE**=**0** suppresses generation of these LOG messages.

When communication errors occur, helpful trace information can be found in the trace data sets, if **LITRACELVL** and **ODTRACELVL** are larger than zero. Tracing can dramatically reduce performance because a lot of data has to be analyzed, formatted and written into the trace file. For normal operations, tracing should be turned off, i.e. both parameters should be set to **0**.

CNSMSGS controls, which LOG messages are echoed to the operator console. All messages, whose code letter is included in the character string value of **CNSMSGS** are written to the console (all messages are always logged, independent of the value of **CNSMSGS**). The additional message types 'O' (ODETTE), 'L' (Linedriver) and + (for long messages) can now also be used.

STATISTICS controls the creation of the statistic log file. **STATISTICS**=**1** creates the file (`$RVSPATH/db/rlstat.log`). It contains a line for each sended or received file with name, date, time and sender/receiver sid. **STATISTICS**=**2** creates the same file, but with extended information (e.g. the file name for the transmission (virtual file name), the file size and command nummbers for `SE`, `SK` or `IE`, `IZ`). **STATISTICS**=**3** is the same as **STATISTICS**=**1**, but routed filetransfer will be logged, too (i.e. **SID** of destination station and SID of source station). **STATISTICS**=**4** is the same as **STATISTICS**=**2**, but routed filetransfer will be logged too. **STATISTICS**=**5** means a detailed output in (`$RVSPATH/db/rlstat.log`). with the new parameters such as file format, state of transmission and number of dial tries. **STATISTICS**=**6** produces a more detailed output about deleted entries (by the user) with the cause of deletion (if specified with `delcmd`). **STATISTICS**=**7** includes all of **6** with information about routing. **STATISTICS**=**0** prevents the creation of this file.

III. Utilities

In this section all useful and important utilities of the rvs® system are described.

9. List of all utilities

This list gives you an overview of all rvs® utilities and there availability on different operating systems.

- `rvsjs` (Windows and UNIX)
- `rvswrdstat` (Windows and UNIX)
- `rvsut2fv` (Windows and UNIX)
- `rvseerp` (Windows and UNIX)
- `rvssce` (Windows and UNIX)
- `rvsap` (Windows, AIX, IRIX and Solaris)
- `rvsrii` (Windows and UNIX)
- `rvsie` (Windows and UNIX)
- `rvsbackup` (only UNIX)
- `rvsrestore` (only UNIX)
- `rvssend` (only UNIX)

9.1. rvsjs

A functionality, installation and configuration of rvsjs will be presented in this chapter.

9.1.1. Introduction: rvsjs as Extension of rvs® Batch Interface

The following steps are needed to perform a send job via rvs® batch interface:

- provide the send file,
- provide the job file (contains send job parameters like address ID, OFTP file name, conversion type),
- call the rvsbat command (see [1]).

This procedure requires that the rvsbat command should be executed on the same machine where rvs® is installed. rvsjs extends the rvs® batch interface to allow creating of send jobs via network. rvsjs forwards job files to rvsbat, that are being stored in a previously configured directory (job directory). If the job directory is visible in a network rvs® may be caused to perform send jobs via network. This constellation is shown in the following illustration:

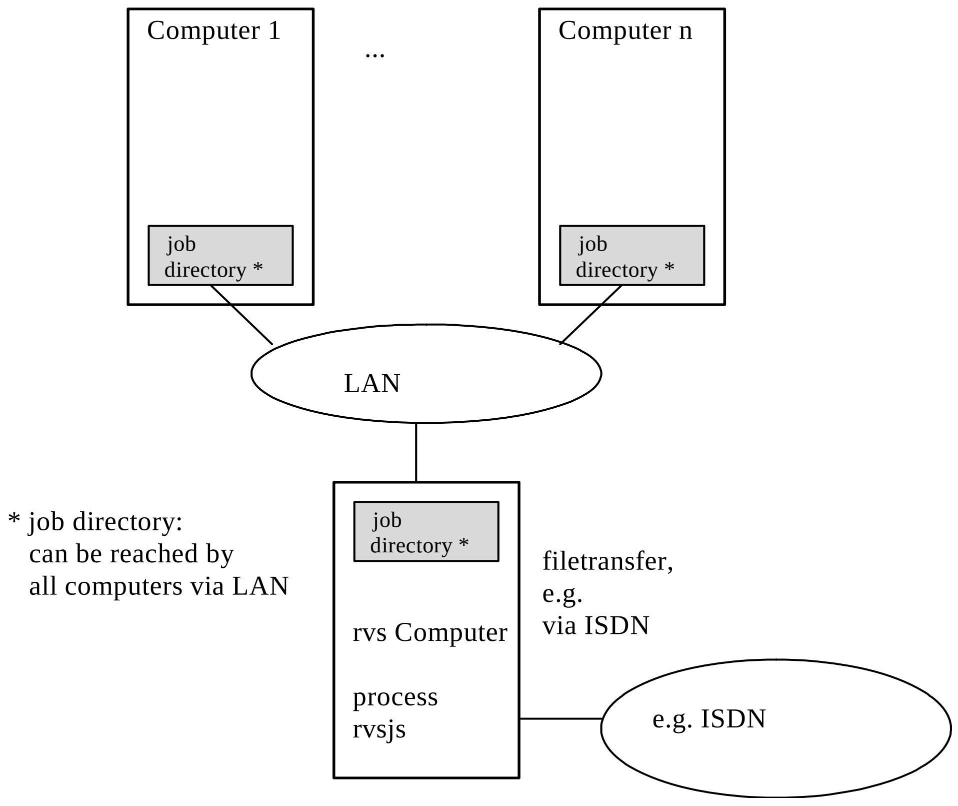


Illustration 1: Sample configuration, creation of rvs send jobs via network directory

From Illustration 1 follows that via network directory rvs® send jobs may be generated by several computers.

The job file can contain the following rvsbat commands:

- create and delete send jobs (`SEND /CREATE`, `SEND /DELETE`)
- create, delete, modify resident receive entries and jobstarts (`RESENTR /CREATE`, `RESENTR /DELETE`, `RESENTR /UPDATE`; `SENDJOB /CREATE`, `SENDJOB /DELETE`, `SENDJOB /UPDATE`)
- modify the station table (`MODST`),
- set and list the rvs® parameters (`SETPARM` and `LISTPARM`).

One job file can list several rvs® commands.

9.1.2. Working Method of rvsjs

rvsjs performs the following steps cyclically:

rvsjs monitors one or several *job directories* to find *job files* for rvs. *Job files* are identified by (configurable) distinctive filenames (e.g. `*.job`).

When rvsjs notices a *job file* it renames the file to avoid further processing as job file. This intermediate filename has the suffix `.js`.

rvsjs launches rvsbat (resp. another configurable *job application*) and hands over the *job file* on the *job application* command line.

After termination of rvsbat (resp. of the *job application*) rvsjs evaluates the return value. If the return value is 0 rvsjs assumes successfull execution and deletes the *job file* `*.js`.

If the return value is different from 0 rvsjs assumes the call failed. In this case rvsjs writes the return value into the *job file* and renames it to `*.err`. After that rvsjs continues with step 1.

The job file is processed immediately after it appeares in the job directory. Therefore it should be written first with another name and after that be renamed to `*.job`.

It is important to consider that the contents of the job file is evaluated by rvs®. This requires the names of value files in the job file refer to the network position of the rvs® computer.

9.1.3. Installation

This chapter decribes the installation of rvsjs on Windows and on UNIX systems.

Windows

The executable file `rvsjs.exe` must be in the directory `$RVSPATH/system`.

The installation of rvsjs as Windows service is performed with the following command:

```
rvsjs -i .
```

With the command

```
rvsjs –i –s
```

rvsjs will be installed as service that is launched automatically when the Operating System is booted.

If rvsjs shall use a certain user account it has to be installed by the following command:

```
rvsjs –i –u <domain name\user name> -x <password>
```

Example:

```
rvsjs –i –u TSYSTEMS\GOR –x gor1?
```

Hints:

- The domain name for the local account is ‚.‘.
- If rvsjs doesn’t work under system account, output to the current desktop isn’t possible. Therefore rvsjs can’t be used to launch applications that require screen output.
- rvsjs and “Samba” directory: From Samba Version 2.0 and above the feature “directory change notification” (necessary for rvsjs) is supported.

UNIX systems:

The executable file `rvsjs.exe` must be in the directory `$RVSPATH/system`, but no separate installation steps are necessary.

9.1.4. Konfiguration

This chapter decribes the configuration of rvsjs via commandline and configuration file `$RVSPATH\init\rvsjs.cnf`.

A directory, which will be checked by rvsjs should be configurated with the following command:

```
rvsjs -p <job directory> [-a <job application>]
[-j <job file mask>] [-g]
```

Parameter	**Description**

`-p <job directory>`	Directory which has to be checked for job files
`-a <job application>`	Optional: Set the complete path of the application, which should be started, when the job file is found. Default: `rvsbat`
`-j <job file mask>`	Optional: Sets the filter of file names, which has to be looked for. The wildcards * (for several pattern) and ? (for one pattern) are supported. Example: `*.send`.
`-g`	Optional: indicates that rvsjs has to create generation files bevor processing the application. Therefore appends rvsjs the found file with suffix `.nnnnn`, where nnnnn is a continuous count started with '00001'. An existing suffix will not be replaced. The generation identifier is always added after suffix. **Example**: `new.send.00001`

Example (Windows):

```
rvsjs –p C:\rvs_out –j *.snd
```

In this example is the job directory `C:\rvs_out` and a job filter mask should end with `snd`. rvsbat will be used as default application. rvsjs should check the directory `C:\rvs_out` and execute the job files ending with `snd`.

Configuration file `$RVSPATH\init\rvsjs.cnf`

After the first configuration step you will find the rvsjs configuration file `rvsjs.cnf` (on Windows system the suffix `cnf` will not be shown) in the directory `$RVSPATH\init`. All configuration steps are held on in this file. This file can also be edited using a text editor.

Windows systems:

The following syntax should be applied in the file `rvsjs.cnf`:

```
<job directory> [-g] <job file mask> <job application>
```

As separater between the parameters blank or tab should be used.

Example:

```
c:\out    –g   *vfs*     sendfile.cmd
```

In this example the directory `C:\out` will be checked for the files with the mask `*vfs*` and if those files are found the job application `sendfile.cmd` will be called. As separator tab was used.

Example for `sendfile.cmd`:

```
SEND /C DSN=C:\gfd\calc007.txt (SID=O43 DSNNEW=calc7)
```

UNIX systems:

This file can have one or more lines for each directory that have to be supervised.

Pro line the following syntax should be applied:

```
<job directory> tab [-g] tab <job file mask> tab <job application>
```

Between the separate options a tab should be used.

If you use the defaults, you should use the defaults for both optional parameters (job file mask and job application) or write them both in the configuration file. It is not possible to use a default for one parameter (e.g. job file mask) and set the second parameter to own value, because the order of the parameters in a line is important.

Note: All configuration changes take effect after start of rvsjs. You can check this file for information about the actual configuration. The actual configuration will be also shown with the command `rvsjs -?`.

9.1.5. Start and Stop

Windows

rvsjs on WINDOWS systems should be started by the command

```
rvsjs –t <update interval in seconds>
```

Parameter `–t` is the time interval for supervising the job directory.

Example:

```
rvsjs -t 60
```

In this example rvsjs will be started and it will check the job directory every 60 seconds.

rvsjs service

The rvsjs service can be started, stopped or configured for automatic start with the operating system by means of the Windows service control template.

On the DOS prompt or in batch programs start is done by

`rvsjs` or by

```
rvsjs –b.
```

`rvsjs` is terminated by

```
rvsjs -e.
```

Hint: rvsjs may also be launched with the rvs® Monitor by adding the instruction

```
system cmd=rvsjs.exe -b
```

to the file `$RVSPATH/init/rdmini.dat`.

UNIX systems

Hint: You should configure the rvsjs configuration file at first, because at start of rvsjs the time interval for supervising the job directory is needed (see chapter 9.1.4).

rvsjs on UNIX systems should be started by the command

```
rvsjs –t <update interval in seconds>
```

It is advisable to add this command with the following command

```
system cmd=“rvsjs -t <update interval in seconds>“
```

to the file `$RVSPATH/init/rdmini.dat` because rvsjs automatically terminates when the rvs® monitor finishes working.

Example:

```
system cmd=“rvsjs -t 60“
```

In this example rvsjs will be started and it will check the job directory from the configuration file `$RVSPATH\init\rvsjs.cnf` every minute (every 60 seconds).

rvsjs on UNIX systems may also be stopped by sending of SIGQUIT:

```
kill –s SIGQUIT <pid>
```

When rvsjs is executed with the option `-c` then it runs in the foreground.

9.1.6. Automatical file sending from supervising directory

rvsjs can also be used for automatical sending of files, which appears in the for rvsjs configured directories. For this purpose rvsjs should be used together with the rvs® utility `rvssce` (see chapter 9.10).

Here is an example:

The rvsjs configuration file `rvsjs.cnf` should be configurated as usual:

Example:

```
c:\out –g *kfg* sendfile.cmd
```

In the directory `c:\out` rvsjs will try to find the files with the mask `*kfg*` and than execute the application `sendfile.cmd`.

In the file `sendfile.cmd` the rvs® send utility `rvssce` should be used.

Example:

```
rvssce -d %1 -s H78
```

In this example `rvssce` should be use to send all files from the directory `c:\out` which applies to the mask from the configuration file to the station `H78`. The value of the parameter after the option –d is the file, which should be sent from the directory `c:\out` to the station `H78`.

9.1.7. Reference

rvsjs in rvsXP

rvsjs -?	show help text and the actual configuration
rvsjs -p <job directory> [-a <job applikation>] [-j <job file mask>] [-g]	Job directory configuration (takes effect after restart); job application, job file mask are optional parameters. Default: rvsbat and .job (see chapter 9.1.4).
rvsjs –t <scantime>	Start rvsjs with time interval for supervising the job directory
rvsjs -r <directory>	remove supervising directory from the liste of job directories (takes effect after restart)
rvsjs -i	Install rvsjs as XP service
rvsjs –i -s	install rvsjs as XP service which starts with Windows
rvsjs –i –u <account> -x <password>	install rvsjs as XP service, under defined account
rvsjs –d	uninstall rvsjs as XP service
rvsjs -b	start rvsjs as service
rvsjs -e	stop rvsjs service
rvsjs -c	starts rvsjs in the console
rvsjs -f	Normally rvsjs does not start if the job directory is not configurated (or not reachable). This option allows rvsjs to start and check periodically if the directory already exists or is available.

rvsjs in rvs X

rvsjs -?	help
rvsjs -p <job directory> [-a <job applikation>] [-j <job file mask>] [-g]	Job directory configuration (takes effect after restart); job application, job file mask are optional parameters. Default: rvsbat and .job (see chapter 9.1.4).
rvsjs -t <update interval in seconds>	Start rvsjs with time interval for

	supervising the job directory
rvsjs -f	Normally rvsjs does not start if the job directory is not configurated (or not reachable). This option allows rvsjs to start and check periodically if the directory already exists or is available.
rvsjs -c	Starts rvsjs in console, not as demon (background process)
rvsjs –m <limit>	rvs load limit (max number of KT commands)
rvsjs –n <number>	Max number of aktive jobs

9.1.8. rvsbat error values

In this chapter we will give a list of the rvsbat error values:

Situation	Error Value
Send file from a job file is missing (value of the parameter 'DSN')	99
No entry 'DSN' (Data Set Name, send file) in the job file	99
No entry 'SID' (Target-ID) in the job file	99
Missing rvs administration rights (e.g. for rvs command 'act')	99
wrong syntax	1
wrong syntax	2
wrong syntax	3

9.2. Write a station table to a backup file (rvswrdstat)

The tool `rvswrdstat` on Windows and UNIX systems gives you a possibility to make a backup file of the station table. The station table should be configurated by the graphical user interface on Windows systems and in the file `$RVSPATH/init/rdstat.dat` on Unix systems.

Usage:

```
   rvswrdstat [-?o]
```

Optional parameters:

-?	help
-o <output file>	Write to the specified output file; without this option the station table will be written to the standard output.

Example:

```
rvswrdstat –o /home/skk/rvs/arcdir/rdstat12.dat
```

The backup file of the station table can be imported into the rvs with the following steps:

UNIX systems:

- Move the old station table `$RVSPATH/init/rdstat.dat` to a safe place.
- Copy the backup file (e.g. `rdstat12.dat`) to the directory `$RVSPATH/init/` and store it under the file name `rdstat.dat`.
- Then modify the rvs database with the command
  `modst`
  in the Operator Console (`rvscns`).

Windows systems:

Use the graphical interface to import stations.

- Open the station table window with the rvsNT Administrator (`View ⇒ Stations`) or rvsXP Administrator (`rvs ⇒ Stations`).
- Execute the command `Edit ⇒ Import Station Table`.

Then `Select file with station definitions` dialog window will open and display, the files which have the name `*.dat` in the rvsNT or rvsXP system directory.

- Search the directory with the station table backup file and select the file which has to be imported.
- Confirm your selection by pressing **Open**.

9.3. Convert U or T File to pseudo F or V Format (`rvsut2fv`)

On many platforms like **PC** or **UNIX** systems, there are no (native) record format files, all data sets are either ASCII or binary files and thus are transmitted as either **T** or **U** format files, respectively, by rvs®.

When you want to send a file to another system that does support fixed and/or variable record format data sets and if you want to make sure that your data will be delivered with a particular record size, you can use `rvsut2fv` to convert your stream file to a (pseudo) record format file that you can transmit specifying **F** or **V** format in the dialog or batch interface. For example, this procedure is necessary if you need to transfer an unstructured file from a UNIX system to an MVS host where they must be stored there in a certain **F** or **V** format.

Remember, however, that automatic code conversion between ASCII and EBCDIC takes only place for **T** format data sets, so that you may have to specify input and output codes when sending text data sets as record files.

Usage

```
rvsut2fv <output file> <F or V>
<record length> <input file> <U or T>
```

All parameters are required:

<output file>	(fully qualified) name of output file; (sub-) directory must exist.
<F or V>	one-character format for output file (**F** or **V**)
<record length>	record length for **F** format, maximum record length for **V** format output file.
<input file>	name of input file.
<U or T>	one-character format specifying whether input file should be interpreted as binary (**U**) or text file (**T**).

For **T** format input files, each line is terminated by carriage return and line feed; each line is converted into one output record. Longer lines are truncated to <record length>; for **F** format output files, shorter lines are padded with blanks.

For **U** format input files, each output record contains <record length> bytes, except for the last one, which may be shorter for V format output files; for **F** format output file, it may contain trailing zeros.

For more information see the User Manual, the chapter about code conversion and the chapter about the rvs® parameters, parameter **VFTYP**.

9.4. Active Panel (`rvsap`)

This new feature of rvs® enables the rvs® administrator to get more information about state and progress of transmission. It is only available for Windows, AIX, IRIX and Solaris.

This program you can start executing

- The command `rvsap` (for UNIX systems)
- `View → Active Lines` menu command in the rvsNT / rvsXPAdministrator (Windows)

The following details are available on this panel:

SID	Station ID
State	state of the transmission at the level of the ODETTE-Protocol
R(R)	I am receiving a file and I am the Initiator (active) of the communication process.
S(S)	I am sending a file and I am the Initiator of the communication process.
R(S)	I am receiving a file and I am the Responder (not active) in the communication process.
S(R)	I am sending a file and I am the Responder in the communication process.
Lyne Type	the type of communication
Process ID	Id for communication process at the operating system level
DSN	Data set Name, the name of the file, which is just in transmission
v(B/s)	Velocity (speed) of the transmission
rate	how much of the transmission has already been done (percentage)
start	the start time of the transmission.

If you have too many busy lines at the same time and you are interested only in some of them, you can define a filter which results in showing you only the lines of interest. In order to define a filter you can define the filter by indicating

- a **SID**: restricts to lines going to this station, * means no restriction by station
- a communication **TYPE**: restricts to lines of this communication type, * means no restriction by communication type.

<F3> function key or an **<ESC>** key exits this panel on UNIX systems.

9.5. Recover Isam Index (`rvsrii`)

The rvs® database is isam (index sequential access method) organized.

In the `$RVSPATH/db/` directory there are two types of files:

```
*.db
```

```
*.idx
```

The `*.db` files are table files and the **`*.idx`** are index files.

Every access to a table file is running over ist `.idx` file. Index files can grow very much. Therefore, they have to be recovered regulary.

```
rvsrii ['/eenvdsn'] ['/lx']
```

All parameters are optional.

- the optional parameter **/e** is used only, if the environment data set is not defined in the **RVSENV** environment variable and not located in the current directory, either.
- the optional parameter **/l** defines the language (**x**) to be used for prompts and messages, default is English.

Example:

```
rvsrii /ld
```

With this command you can call rvsrii with German language for prompts and messages.

This utility may be started while rvs is running.

You should invoke this command from the file

```
$RVSPATH/init/rdmini.dat
```

by entering `OPCMD` operator command

```
OPCMD cmd='system cmd="nohup rvsrii > /dev/null
&"' time=02:00:00 repeat=24:00:00
```

This command enables database index cleanup every day at 2 o'clock.

If you are forced to use the rvskill command, we strongly recommend to call the rvsrii utility afterwards to avoid the possible damaging of the database.

9.6. rvs® Information Entry (`rvsie`)

`rvsie` (the rvs® Information Entry recovery tool), can be very helpful, if your database get damaged. `rvsie` make it possible for you to recover all the information entries, which are not successfully completed, because exactly such uncompleted rvs® commands can be lost, if the database gets damaged.

What kind of information entries is it possible to recover with `rvsie`?

- All `SE`s of files, which are not completely transmitted
- All `IE`s of files, which are completely received, but are destinated for routing
- All `QS`s, which are still to be sent

Usage

```
rvsie [-fvsdriomtqanl]
```

-f	local filename (**DSNLOCAL**)
-v	virtual filename (**VDSN**)
-s	SID of sender (**SIDSENDER**)
-d	SID of destination (**SIDDEST**)
-r	record format (**RECFM**)
-i	input character code (**A**=ASCII or **E**=EBCDIC)
-o	output character code (**A**=ASCII or **E**=EBCDIC)
-m	max. record length (**MAXRECL**)
-t	time (**DTAVAIL**)
-q	create End-to-End-Response (**EERP**)
-a	immediate call to partner (only **EERP** feature)
-n	optional neighbour SID (only **EERP** feature)
-l	print list of entries in rvs® database (input for `rvsie`)
-c	iDaysBefore: print input for `rvsbat` to cancel send entries

To recover your damaged database with `rvsie`, it is neccessary to do the following steps:

- With command
  `rvsie –l`
  you can list all information entries from the database, which are not yet executed. For every information entry the corresponding call of `rvsie` is generated.
- The next step is to redirect this output into a command file
  Example for **Windows** : `rvsie -l > restore.bat`
  Example for **UNIX**: `rvsie -l > restore`
- Now you have to delete the damaged database with the command:
  `rvsdbdel`
- And to create a new one, the command:
  `rvsidb lid`
  where **lid** is replaced by your local station ID.
- After that you have to execute the generated command file
  Example for **Windows** : `restore.bat`
  Example for **UNIX**: `sh restore`.

If there are any information entries in the output file which are no longer important for you, you can first edit this file in order to delete the unnecessary information entries before executing. By execution of the output file the command `rvsie` is called so many times, as you have unexecuted information entries. So, you do not have to call it manually for every entry.

9.7. Backup of the rvs® data (`rvsbackup`)

It is very important to practise a regular backup of the rvs data in order to beeing prepared for e.g. data loss after power failure. This tool helps you to do this task automatically.

`rvsbackup` checks

- that no other rvs process is running
- that the selected archive directory is existing

`rvsbackup` creates

- a dump of the rvs® database (`rvsdbdump.log`)
- a file `rvsenv.var` which contains a value of **$RVSENV** (**Example**: `$RVSPATH/rvsenv.dat`)

`rvsbackup` makes copies of

- all db log files (`rlog.log`, `rlstat.log`, `rldb.log`)
- all files from `$RVSPATH/temp` directory
- the configuration file (`$RVSPATH/rvsenv.dat`)

- all files from `$RVSPATH/init` directory (`rdkey.dat`, `rdmini.dat`, `rdstat.dat`)

Usage

```
rvsbackup [options]
```

-?	usage
-a	all backup steps in **$ARCDIR**
-e	save **$RVSENV**
-b	dump database
-x	delete dblog after successful dump
-s	check if rvs is stopped
-c	backup copy (`rlog.log`, `rldb.log`, `rlstat.log`, `init/*`, `rvsenv.dat`, `temp/*`)
-r	call `prervsbackupext`
-o	call `postrvsbackupext`
-n	Create and use new subdirectory in **$ARCDIR**
-d dir	save in directory `dir`

Examples:

```
rvsbackup –a
```

This command performs **all** backup steps into archive directory **$ARCDIR**.

```
rvsbackup –a –d /home/tmp/backup
```

This command performs all backup steps into `/home/tmp/backup` directory.

There is the possibilty to extend the backup process by your own installation-specific scripts. The script `prervsbackupext`, if exists, will be called after the checks before any access to the database.

The `postrvsbackupext`, if exists will be called after all the backup steps are done. The both scripts are searched in the system directory.

If you have made a backup with this tool, you can restore the saved state of the rvs® system by the tool `rvsrestore`.

9.8. Restore of the rvs® data (`rvsrestore`)

This tool serves to restore a former state of the rvs® system from a backup created by the tool `rvsbackup`.

`rvsrestore` checks

- that no other rvs process is running

`rvsrestore` copies the backup

- rvs® enviroment file (`$RVSPATH/rvsenv.dat`)
- all files from the `$RVSPATH/init` directory (`rdkey.dat`, `rdstat.dat`, `rdmini.dat`)
- db log files (`rlog.log`, `rlstat.log`, `rldb.log`)
- all files from the `$RVSPATH/temp` directory

`rvsrestore` delets the old database and creates a new one and fills it with old data from the backup.

Usage

```
rvsrestore [options]
```

-?	usage
-a	all restore steps
-s	check if rvs® is stopped
-e	copy $**RVSENV** (`rvsenv.dat`)
-k	copy `rdkey.dat`
-r	copy `rdstat.dat`
-i	copy `rdmini.dat`
-l	copy `rlog.log`
-o	copy `rlstat.log`
-d	restore database from dump file
-m	copy files from `temp` directory
-b	restore database from `rldb.log`
-x	delete `rldb.log` after successful restore of database
-f dir	(from) archive directory

Example:

```
rvsrestore –a –f /home/skk/rvs/arcdir
```

This command performs all restore steps from archive directory (`/home/skk/rvs/arcdir`).

9.9. rvs® End-to-End Response (`rvseerp`)

EERP (End-to-End Response) is an important service of the ODETTE protocol. It should be send from the final destination to the originator of a transfer file. A file is regarded as completely sent, only if its **EERP** is delivered, too. So, you may have unfinished commands in the rvs® database, if your partner doesn't send you **EERP**, or in the another direction, if you are not in possibility to deliver the **EERP** to your partner.

This tool `rvseerp` enables you to list und handle

send jobs with not yet received **EERP**

receive jobs for which the **EERP** could not be send

In the first case the SK commands remain in status „pending" and in the second case the same holds about the QS. For more information see 2.1.1 "Monitor: Basic Characteristics".

Usage

```
rvseerp [-?lceqsnt]
```

This program has 3 basic features:

- list pending SK/QS command entries (Option **–l**)
- end pending SK command entries (Option **–c**)
- end pending QS command entries (Option **–e**)

The output of the first feature (Option **–l**) contains detailed information about command entry in status „pending" and a `rvseerp` command, that may be used to finish this command entry. The `rvseerp` command is written as a comment (rem for **NT**, # for **UNIX**).

The additional options for the -l option are:

- -q
- -s

- -t

Example:

```
rvseerp –l –s –t 24
```

lists all pending `SKs` older than 24 hours.

You may redirekt the output of a `rvseerp –l` into a file and edit it removing the comment markers in the commands, where `rvseerp` is called with option `–c` or `–e` in order to cancel the pending `SKs` and `QSs` , respectively (the second and the third basic feature of `rvseerp`).

Example:

```
rvseerp –l
```

shows the detailed list of pending command entries with command members.

`rvseerp –l > output.bat` (for **NT**)

`rvseerp –l > output` (for **UNIX**)

redirects the output of the `rvseerp –l` command in an output file.

Now, you can edit the output file, remove the comment markers for commands which you would like to execute, and invoke the edited file.

`output.bat` (for **NT**)

`sh output` (for **UNIX**)

The second and the third basic feature can be called separately:

Examples:

`rvseerp –c –n XXXXXX` (for `SKs`)

`rvseerp –e –n XXXXXX` (for `QSs`)

`XXXXXX` is the command number from a rvs® database generated with `rvseerp -l`.

You should use this tool very carefully and seldom because it bypasses the standard communication protocol (ODETTE). The better solution is to use the ODETTE Parameter **EERP_IN** or **EERP_OUT** in your stationtable.

Note: You can also release or delete a EERP with `rvseerp`. Please, read the User Manual, chapter 3.1.7 for more information.

9.10. rvssce

The tool `rvssce` offers starting from the version 2.05 of rvsX and rvsXP an additional possibility of sending a file to the partner. All send parameters, which are available in rvsX or rvsXP, `rvsbat` and `rvscal`, are usable here with same characteristics. With `rvssce` you can also query the status of a command from the rvs® database and return this information into a XML file in order to process it.

Sending a file

Syntax:

```
rvssce -d <file name> -s <StationID>
[-uvtlIoODFTSVMYC]
```

Mandatory Parameters

-d	Name of the file to be sent
-s	SID of the target station

Optional Parameters

-u ID of the local user

-v virtual file name for the transmission, maximum length 26 characters

-V Text files can also be sent in the format **F** (fixed) or **V** (variable), without conversion by the utility `rvsut2fv` (see 9.1).

VFTYP=T means, that your file is to be sent without conversion by `rvsut2fv`. In this case you have to use the parameters **MAXRECL** and **FORMAT** to achieve the same result as with `rvsut2fv`. See examples in chapter 11.4.

VFTYP=V means, that the file to be sent was already converted by `rvsut2fv`. Now you have to use only the parameter **FORMAT** without the parameter **MAXRECL**. See examples in chapter 11.4.

-D Indication whether the file is to remain or to be deleted after successful sending:

- **K** to keep data after sending (default).
- **D** to delete data after sending.

-l Label, Name of group of serialized send requests.

-F Format of the file to be sent: **T**=text, **U**=unstructured (binary), **F**=fixed, **V**=variable record length.

default: record format of local data set

(**U** for rvsNT, rvsX, **F** for rvs400).

-T earliest time when send request may be performed; may be: **H**=hold, **N**=now (default), or an explicit time in the format **YYYY/MM/DD HH:MM:SS**.

-S Serialization; if set to **Y** (=yes), the send requests will sent in the order you have created them (see also **LABEL**). The next request will only be sent if the previous is completely finished.

-I code of local data set (**A**=ASCII, **E**=EBCDIC);

Default: local code.

-O desired code of data set at receiver; output code (**A**=ASCII or **E**=EBCDIC)

-t Path of the own conversion table.

-M Record length; same as the parameter **MAXRECL** (see chapter 11.4.

-Y Encryption;

- if **Y** (Yes) the file should be compressed before sending
- if **N** (No) the file should not be compressed before sending.

-C Compression;

- if **Y** (Yes) the file should be compressed before sending
- if **N** (No) the file should not be compressed before sending.

-o Output in XML file

-f Odette ID of originator

-i Odette ID of destination station

Example:

```
rvssce -d /home/skk/out/test2.txt -s D43
```

The file `/home/skk/out/test2.txt` should be sent to the station `D43`.

There is also a possibility to write the `rvssce` command in the file `$RVSPATH/init/rdmini.dat`, so it can be executed e.g. periodically.

Example:

```
opcmd        CMD="system        CMD='rvssce        -d
D:\Testdaten\test42.txt -s DL3'" repeat=04:00:00
```

In this example a file `test42.txt` will be sent every 4 hours to the partner station `DL3`.

Inquiry about the Status of a rvs® Command

Syntax:

```
rvssce -c <Command number> -o <XML file >
```

Example:

```
rvssce -c 89 -o /home/skk/out/ausgabe.xml
```

This command line writes the status of the command number 89 into the file `/home/skk/out/ausgabe.xml`. The status of the command 89 is e.g. terminated `(<ended>)`.

Note: However the monitor parameter **SSCREATE** (see the User Manual, chapter "The rvs® Parameters' Overview", parameter **SSCREATE**) must be set to 1. Only in this case the information about the successfully completed commands are maintained in the statistics database table SS. This does not cause performance problems with the rvs® database.

In case of success the return code of `rvssce` is 0. Follows some of possible error return cases:

```
#define RC_INVALID_ARG                        11
#define ERROR_LOAD_LIBRARY                    12
```

```
#define ERROR_GETPROCESSADDRESS           13
#define ERROR_CANNNOT_OPEN_XMLOUT         14
#define ERROR_CANNOT_CREATE_SE            15
```

9.11. rvscheckdb

The `rvscheckdb` program analyzes the rvs® database tables for potential inconsistencies and can correct these if necessary. Inconsistencies can occur when file transmission/reception was suddenly interrupted, e.g. in case of a faulty communication connection.

Syntax:

```
rvscheckdb [-?| –o <file name>| -T <table name>
-C <command number> -d]
```

Options:

-?	Help
-o <file name>	Output file: You can pipe the output of the `rvscheckdb` program to an output file.
-T <table name>	Table of the rvs® database from which the commands are to be deleted.
-C <command number>	Number of the command to be deleted.
-d	Specify the action to be performed with a command: d (delete)

Usage:

Type the following command at the command prompt of the respective operating system:

```
rvscheckdb -o <file name>
```

Example (Windows):

```
rvscheckdb –o C:\dboutput
```

The above example results in a summary of the database check output to the command prompt of the operating system.

In addition, `rvscheckdb` generates two text files showing the database check results in more detail:

- `<file name>.ok`: containing a list of all consistent database entries,
- `<file name>.err`: containing a list of all inconsistent database entries,

and a delete script. Depending on the operating system used, this script bears one of the following names:

- `<file name>.sh:` for Unix operating systems, or
- `<file name>.bat:` for Windows operating systems.

Example:

The `rvscheckdb –o C:\dboutput` command created the following files:

`dboutput.ok`, `dboutput.err` and `dboutput.bat`.

The user can now adapt the delete script as desired with a text editor. During “adapting” you will remove comment characters preceding those delete commands that you wish to have executed on the database.

In Unix operating systems you must delete the “#” comment character preceding the delete commands to be executed.

In a Windows operating system you must delete the “REM echo” comment preceding the delete commands to be executed.

The following example shows a faulty send job entry contained in the delete script.

Example: `dboutput.bat`

```
REM echo Sendjob 1400764 inconsistent!
REM echo <UNKNOWN> was to be sent at 0 from
RVSFARM to LIN4 as
REM echo LOOP.TENNIS.U000.QPRQJQ.
REM echo DB entries for deleting:
REM echo SE 1400764 KT 1400764 ET 1400764

REM echo rvscheckdb -T SE -C 1400764 -d
```

The upper lines in the example are for explanation. You must **NOT** delete the comment (`REM echo`) preceding these lines.

The second and third line give an overview of the send/receive job using available job information in the following pattern:

`<file name>` was to be sent at `<time>` from `<sender>` to `<recipient>` as `<virtual file name>`.

Where the database does not contain any information on the job, the `<UNKNOWN>` or `0` (for the time) default is used.

The last three commented lines of the above example are the actual delete commands the `rvscheckdb` program can interpret and execute.

Note: Depending on the completeness the number of delete commands for each entry can vary. Remove the comments for only these lines if required!

After editing and saving the delete script the user can execute it from the operating system’s command line.

9.12. Send a Data Set (`rvssend`) only for UNIX

`rvssend` lets you send a text file to a rvs® station:

```
rvssend local_filename remote_filename sid
```

This command is a simple shell script in the system directory. It can be used to test a connection. The following example sends the system profile as "HELLO" to the station "ABC":

```
rvssend /etc/profile HELLO ABC
```

IV. rvs® Interfaces

The rvs® interfaces `rvsbat` and `rvscal` are described in this part.

10. How to work with rvs® Batch Interface and rvs® C-CAL Interface

This chapter describes how to start the two interfaces and which parameters you can use. Furthermore, a description of the corresponding global commands is given. The syntax of the commands is explained as well as the prototype of the function `rvscal()`.

Note: For all tasks (sending and receiving data files as well as administration of rvs®) you have to do with rvs® you can either

- write commands into a text file and execute them by the command line interface (`rvsbat`)
- execute commands by the function `rvscal()` of the C-CAL interface
- use several dedicated C- functions of the C-CAL interface (each command has own function)

The string commands of `rvscal()` are identical to those useable for `rvsbat`. You can read about them in chapter 11 "Description of Commands".

On the other hand the dedicated functions are on a lower abstraction level thus providing more control and flexibility. They are described in chapter 12 "How to Work with rvs® C-CAL Interface".

10.1. Start the rvs® Batch Interface (rvsbat)

The batch interface can be invoked as specified below:

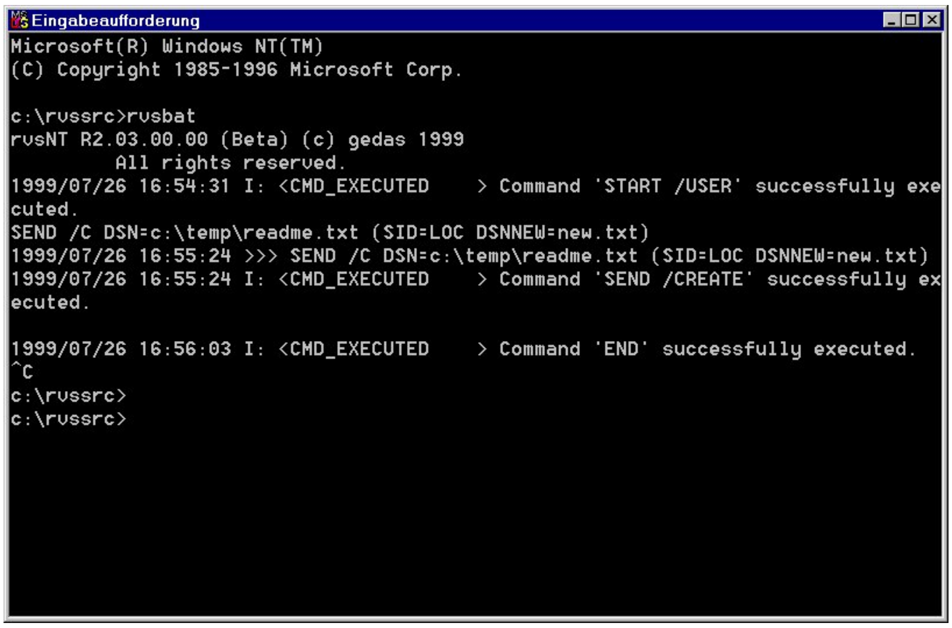
```
rvsbat [/c] [/e<env.dsn>] [/i<cmd input file>]
[/l<language>] [/q]
```

Note: The invocation and parameter passing mechanism are dependend on the target operating system.

The command line parameters have the following meaning:

- **/c**: continue with `rvsbat` after an error occured during execution of a utility command. By default, `rvsbat` will terminate after an error.
- **/e**: use non-default environment data set
- **/i**`cmdfile`: do not read commands from `stdin` but from `cmdfile`. The command input file may contain the following elements:
– Comment lines (starting with *****)
– Commands (may extend over several lines by specifying **+** as the last character in the line to be continued)
- **/l**`language`: use message language given by character language
- **/q** execute user commands in quiet mode, i.e. do not echo them to standard output; feedback about success or failure of the operation will still be provided.

The following picture shows an example for **Windows NT**:



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

c:\rvssrc>rvsbat
rvsNT R2.03.00.00 (Beta) (c) gedas 1999
         All rights reserved.
1999/07/26 16:54:31 I: <CMD_EXECUTED    > Command 'START /USER' successfully exe
cuted.
SEND /C DSN=c:\temp\readme.txt (SID=LOC DSNNEW=new.txt)
1999/07/26 16:55:24 >>> SEND /C DSN=c:\temp\readme.txt (SID=LOC DSNNEW=new.txt)
1999/07/26 16:55:24 I: <CMD_EXECUTED    > Command 'SEND /CREATE' successfully ex
ecuted.

1999/07/26 16:56:03 I: <CMD_EXECUTED    > Command 'END' successfully executed.
^C
c:\rvssrc>
c:\rvssrc>
```

Handling of Commands

Commands can be written into the <cmdfile> or can be entered when calling `rvsbat`.

In both cases, the syntax of the command strings is identical. An example is given in the picture above:

```
SEND /C DSN=c:\temp\readme.txt (SID=LOC DSNNEW=new.txt)
```

For each command read from command input, a message is written to **stdout** which gives information about success or failure of processing the command.

If `rvsbat` is started without input of a command file, it can be breaked by **<STRG> C**.

10.2. How to use the C-CAL Interface (rvscal())

The C-CAL interface is an Application Program Interface (API) for rvs®. The following sections describe how to use the C-language functions that may be linked into an application program to execute rvs® utility commands.

10.2.1. How to compile and link the C-CAL Interface for rvsNT

This implementation of rvs® C-CAL Interface is tested and written in Microsoft Visual C++. The prototype definitions are made in the `rvscal.h` header file. That is what has to be included in the application source file:

- `#include rvscal.h`

There is another header file `rixstd.h` needed but not included. Both are installed in the `sample` directory of rvs®.

It is recommended to use: **Run-Time Linking** to link the rvs® C-CAL Interface. Therefore the application must issue the following calls:

- `LOAD_RVSCAL_DLL(HANDLE &hlib)`. There is a macro to load the correct version of dynamic load library (`RVSCALL.DLL`) of rvs®. This macro calls the function `LoadLibrary()` indirectly and loads the actual version of the rvs® library. The parameter `&hlib` is the address of the handle for the library. We recommend to use this macro to be compatible to future versions of rvs® library.
- `GetProcAddress()` to specify rvs® C-CAL Interface on all desired entry points to the dynamic link library (`DLL`) such as `rvsCreateSendEntry()` etc.
- `FreeLibrary()` issue the `FreeLibrary()` call when the rvs® C-CAL Interface functions are no longer needed

Compiler and linker options:

- **Packing:** Structures of rvs® C-CAL Interface are packed on 4 byte boundaries. Therefore the application has to use the compiler option `/Zp4`.
- **char type**: rvs® C-CAL Interface uses the compiler option `/J` to change the default char type from `signed char` to `unsigned char`.

Library Files

Two dynamic link libraries are delivered which are needed to exist during run-time of the applikation using rvs® C-CAL Interface:

- `RVSCAL.DLL` loaded by the application program
- `RVSCAL22.DLL` loaded during run-time of `RVSCAL.DLL`, must not be loaded by the application.

Example:

```
#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",
GetLastError());
            return;
        }

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

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

10.2.2. Usage of the C-CAL Interface for UNIX and OS/400

To use the C-CAL Interface it is necessary to link the following libraries

- `rpulib.a` for **UNIX** systems (static link). You can find this library in the directory `$RVSPATH/system`.
- `rvscal` for **OS/400** (service program). This service program is in the library `RVS_SYSTEM`.

Then the user defined C program which uses this library has to include the header file `rvscal.h` into the C source code.

UNIX:

In the directory `$RVSPATH/samples` are some examples how to use the C-CAL Interface. The typical usage of the C-CAL Interface is demonstrated in the program `$RVSPATH/system/rvscd`. The C source code for this program you can find in the directory `$RVSPATH/samples/s_rvscd.c`.

In this program you can see:

- how to put entries of station table file `(&RVSPATH/rdstat.dat)` to rvs® database (function `ModifyStationTable`(`char *stationfile`)).
- how to list values of the rvs® parameters (function: `ListParameter(char *parm)`).
- how to send a file (function: `Send(char *dsn, char *dsnnew, char *sid)`).
- how to show station table entries (function: `ListStations()`).
- how to print help text (function `help()`).

In the directory `$RVSPATH/samples` is the make file `s_make.cd` for building the demo program `rvscd`.

10.2.3. Header File `rvscal.h`

This chapter describes how to define a macro and explains a prototype of `rvscal`.

Macro definitions

storage class definitions:

```
#if !defined _LITYPE_INCLUDED
typedef char            STRING ;
# if defined OS400
   typedef int            SINT;
# else
   typedef short          SINT;
# endif
#endif
```

```
#include "rixstd.h"

#ifdef NT
#  define PROCDEF extern  __declspec(dllimport)
#  define PROCKEYW


#elif  _AFXDLL
#  define PROCDEF  extern "C"
#  define PROCKEYW

#else
#  define PROCDEF
#  define PROCKEYW
#endif
```

return codes:

```
#define RVSCAL_OK                              0
#define RVSCAL_END_FETCH -                     1
#define RVSCAL_CLOSE_FAILED                   -2
#define RVSCAL_OPEN_FAILED                    -3
#define RVSCAL_ERROR_CREATESEND               -4
#define RVSCAL_PARAMETER_CHECK                -5
#define RVSCAL_DBERROR                        -7
#define RVSCAL_INVALID_NAME                   12
#define RVSCAL_INVALID_SID                    13
#define RVSCAL_RE_NOT_FOUND                   14
#define RVSCAL_NEITHER_X_NOR_ISDN             15
#define RVSCAL_INVALID_DSN                    16
#define RVSCAL_INVALID_JOB                    17
#define RVSCAL_NOT_PRIVILEGED                 18
#define RVSCAL_DUPLICATE_RE                   19
#define RVSCAL_INVALID_UID                    20
#define RVSCAL_INVALID_OWN_PRIV               21
#define RVSCAL_INVALID_USER                   22
#define RVSCAL_DUPLICATE_USER                 23
#define RVSCAL_NEIGHBOR_STATION               24
#define RVSCAL_RC_WAKE_FAILED                 25
#define RVSCAL_WRITE_INTO_PIPE                26
#define RVSCAL_DSN_NOT_EXIST                  27
#define RVSCAL_ENVIRONMENT_NOT_EXIST          28
#define RVSCAL_DUPLICATE_JS                   29
#define RVSCAL_JS_NOT_FOUND                   30
#define RVSCAL_ERROR_GETVERSION               31
#define RVSCAL_ERROR_MALLOC_INFOELEMENT 32
#define RVSCAL_ERROR_MALLOC_SIDELEMENT  33
#define RVSCAL_INTERNAL_ERROR                 99

#define RVSCAL_NOCMD RVSCAL_END_FETCH
```

others:

```
#define RVSCAL_L_ACCT              RVSCAL_L_USID
#define RVSCAL_L_C1                L_C1
#define RVSCAL_L_DBVERSION         L_DBVERSION
```

```
#define RVSCAL_L_DSN            255
#define RVSCAL_L_DT             L_DT
#define RVSCAL_L_JOBID          L_JOBID
#define RVSCAL_L_KS             L_KS
#define RVSCAL_L_ODETTEID       L_ODETTEID
#define RVSCAL_L_LANG           L_LANG
#define RVSCAL_L_LU62_NETID     L_LU62_NETID
#define RVSCAL_L_LU62_LUNAME    L_LU62_LUNAME
#define RVSCAL_L_LU62_TPNAME    L_LU62_TPNAME
#define RVSCAL_L_LU62_UID       L_LU62_UID
#define RVSCAL_L_LU62_PASSW     L_LU62_PASSW
#define RVSCAL_L_LU62_PROF      L_LU62_PROF
#define RVSCAL_L_LU62_MODE      L_LU62_MODE
#define RVSCAL_L_LU62_PIP_DAT   L_LU62_PIP_DAT
#define RVSCAL_L_LX_NAME        L_LX_NAME
#define RVSCAL_L_LX_VALUE       L_LX_VALUE
#define RVSCAL_L_LOGMSG         L_LOGMSG
#define RVSCAL_L_NETID          RVSCAL_L_ODETTEID
#define RVSCAL_L_ERRMSG  512
#define RVSCAL_L_ODETTEID       L_ODETTEID
#define RVSCAL_L_OFTP_EERP      L_OFTP_EERP
#define RVSCAL_L_OFTP_USERD     L_OFTP_USERD
#define RVSCAL_L_OPSW           L_OPSW
#define RVSCAL_L_PARM_NAME      L_PARM_NAME
#define RVSCAL_L_PARM_VAL       L_PARM_VAL
#define RVSCAL_L_PHONE          L_PHONE
#define RVSCAL_L_PIPEBUFFER     20000
#define RVSCAL_L_PRIORITY       L_PRIORITY
#define RVSCAL_L_RECMNT         L_RECMNT
#define RVSCAL_L_SEUSRLABEL     L_SEUSRLABEL
#define RVSCAL_L_STATID         L_STATID
#define RVSCAL_L_STATNAME       L_STATNAME
#define RVSCAL_L_TCPIP_ADDR     L_TCPIP_ADDR
#define RVSCAL_L_TCPIP_SEC      L_TCPIP_SEC
#define RVSCAL_L_USID           L_USID
#define RVSCAL_L_USERNAME       L_USERNAME
#define RVSCAL_L_VDSN           L_VDSN
#define RVSCAL_L_X25_ADDR       L_X25_ADDR
#define RVSCAL_L_X25_ALIAS      L_X25_ALIAS
#define RVSCAL_L_X25_CUG        L_X25_CUG
#define RVSCAL_L_X25_FAC        L_X25_FAC
#define RVSCAL_L_X25_LINK       L_X25_LINK
#define RVSCAL_L_X25_SUBADDR    L_X25_SUBADDR
#define RVSCAL_L_X25_USDTA      L_X25_USDTA
#define RVSCAL_L_X25_ISDNNO     L_X25_ISDNNO
#define RVSCAL_L_X25_TIMEOUT    L_X25_TIMEOUT

#define SID_START          " "
```

Prototype of function `rvscal`

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

Note: parameter list has been designed in this way in order to be callable by other programming languages, e.g. FORTRAN, as well.

Description of Parameters

RETURNVALUE	(int) =**0**, if `rvscal` succeeded (although the message text may have been truncated), different from **0**, if any failure occured. There are two types of error codes: • errors detected by the command parser are due to an invalid command syntax. The error codes are in the range between 1 and 7. A detailed description is given in "Messages and Codes". • Any other error occured while rvs® tries to execute the specified command. The message string returned via **S_MSG** contains a description of the error reason.
S_CMD	(char [], input) pointer to character array with command string; the array size must be at least ***P_L_CMD+1**
P_L_CMD	(int *, input) length of command string; **S_CMD** must be zero-terminated, if ***P_L_CMD** is larger than length of command string
P_C_MSGLVL	(char *, output) returns the message level; may be **I** (informal), **W** (warning), **E** (error), **S** (severe)
S_MSG	(char [], update) pointer to a character string of length at least ***P_L_MSG+1** ; the character string receives a message from `rvscal` which gives information about the success of calling `rvscal`
P_L_MSG	(int *, update) on input, ***P_L_MSG+1** is the size of the buffer used to receive the message from `rvscal`; on output, ***P_L_MSG** gives the actual length

	of the message text. This value does not include the zero-terminator, but **S_MSG** is zero-terminated.
P_CMDID	(long *, output)
	the **CMDID** (command number) the command which was processed (see table below).

The following commands (`s_cmd`) are supported:

`start`	start rvs® utility session
`end`	terminate session
`send`	create, modify (hold or release) or delete a send request
`print`	print documentation about rvs®
`receiver`	delete a single receiver from send request
`resentr`	create, update or delete a resident receive entry
`sendjob`	create, update or delete a entry for starting jobs after send attempts
`user`	create, modify or delete entry in rvs® user table
`activate`	activate a rvs® station
`modst`	modify station table (read a station table file)
`delst`	delete a station table entry
`listparm`	list a rvs® parameter
`setparm`	set a rvs® parameter

The **CMDID** returned by `rvscal` has the following meaning depending on the command passed to `rvscal`:

Command	**meaning of CMDID**
`SEND /CREATE`	number of created send entry
`SEND /DELETE`	number of send entry being deleted[1]
`SEND /HOLD`	number of send entry being held (see footnote below)
`SEND /RELEASE`	number of send entry being released (see footnote below)

[1] If the send entry could not be identified uniquely by the specified parameters, the number of the first matching command is returned.

Command	meaning of CMDID
`RESENTR /CREATE`	created resident receive entry
`RESENTR /DELETE`	deleted resident receive entry
`RESENTR /UPDATE`	number of new resident receive entry
`SENDJOB /CREATE`	created jobstart after send attempt entry
`SENDJOB /DELETE`	deleted jobstart after send attempt entry
`SENDJOB /UPDATE`	number of new jobstart after send attempt entry
`RECEIVER /DELETE`	number of corresponding send entry
`LISTPARM`	not equal **0**: invalid parameter
`SETPARM`	not equal **0**: invalid parameter
`MODST`	not equal **0**: error occured while loading station table
`any other command`	**0**

11. Description of Commands

This section describes the syntax and the set of valid commands used either by the rvs® C-CAL Interface (`rvscal`) or the rvs® batch interface (`rvsbat`).

The examples given below, use two syntactical extensions only available with the batch interface:

- comment lines start with a ***** in column **1**,
- commands can be continued in the next line by specifying a **+** as the last character of the line to be continued

11.1. Syntax of Command Strings

A utility command must follow the syntax rules specified below:

- It consists of
  – a command verb,
  – an optional command qualifier like `/CREATE, /DELETE`, etc. The qualifier may start either with **/** or **-** and may be abbreviated to a single letter
  – values of non-repeatable parameters specified as <parameter name>=<parameter value> . There must not be blanks around the = sign.
  – values of repeatable parameters (`send` command only); they follow the same syntax rules as those for non-repeatable parameters. A group of repeatable parameters is enclosed in parentheses. There may be arbitrary many groups of repeatable parameters.
- Command verb, qualifiers and parameter names may be specified in uppercase, lowercase, or mixed case.
- Parameter values are converted to uppercase, if not protected by single or double quotes.
  If a parameter value is to contain the protecting character itself, the protecting character must be specified twice. E.g. the following parameter specifications are equivalent:

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

  Non-protected parameter values may contain any alphanumeric character. They may not contain a set of special characters, e.g. blanks, single or double quotes, and parentheses.
- Different parameter specifications must be delimited by at least one blank.

11.2. Command `START`

Function:

- Start a session using rvs® utilities
- Open rvs® database,
- Check whether current user is allowed to use the rvs® utilities
- `start /USER` is issued implicitely by the C-CAL Interface, if not issued explicitely. It is always performed implicitely when the batch interface starts execution.

Qualifiers:

`/USER` (default) start session for rvs® user

Parameters:

RVSENV (optional) name of rvs® environment data set

Examples:

Use environment data set in (rvs400 library) **RVS_NEW/DAT(RVSENV)**

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

11.3. Command `END`

Function:

- Terminate session using rvs® utilities
- Close rvs® database
- END must be issued when the C-CAL Interface is used. It is performed implicitely by the batch interface.

Qualifiers:

none

Parameters:

none

Examples:

```
END
```

11.4. Command `SEND`

Function:

- Create
- modify or
- delete

a send request to send a local data set to another rvs® node

Qualifiers:

`/CREATE`	(default) create a send request
`/DELETE`	delete a send request
`/HOLD`	set pending send request into `held'-state
`/RELEASE`	release send request, which has been put into `held'-prior

`SEND /CREATE` parameters:

DSN	(required) name of local data set to be sent, the complete path
CODEIN	(optional) code of local data set (**A**=ASCII, **E**=EBCDIC);
	default: local code
DISP	(optional) disposition for local data set after send request has ended successfully: **K**=keep (default), **D**=delete
FORMAT	(optional) format used for transfer via ODETTE:
	T=text, **U**=unstructured (binary), **F**=fixed, **V**=variable record length;
	default: record format of local data set
	(**U** for rvsNT, rvsX and rvs2, **F** for rvs400)
ACCOUNT	accounting number or code
INITTIME	(optional) earliest time when send request may be performed; may be: **H**=hold, **N**=now (default), or an explicit time in the format **YYYY/MM/DD HH:MM** (**Example**: 2004/09/04 10:43)
LABEL	(optional) user label (up to 20 characters) used to serialize on a preceeding send request (if **SERIAL**=**Y**) or which can be used for serialization by a subsequent send request

SERIAL (optional) if set to **Y** (=yes), the send requests will sent in the order you have created them (see also **LABEL**). The next request will only be sent if the previous is completely finished.

VFTYP (optional) Text files can also be sent in the format **F** (fixed) or **V** (variable), without conversion by the utility `rvsut2fv` (see 9.3).

VFTYP=T means, that a file should be sent without conversion by `rvsut2fv`. Text file means: only ASCII characters are allowed; the record length is the length of a record without line break (CR/LF for Windows systems and LF for UNIX systems).

In this case (**VFTYP=T**) you have to use the parameters **MAXRECL** and **FORMAT** to achieve the same result as with `rvsut2fv`.

Example: If you want to send a file in format **F** with a record length 80, you should create a file containing only ASCII characters with a record length 80 (without line break). The following parameters should be used: VFTYP=T, FORMAT=F, MAXRECL=80. For the files with FORMAT=V (the record lengths are different), the MAXRECL parameter indicates the record with the maximal length.

VFTYP=V (variable) means, that the file to be sent was already converted by `rvsut2fv`. Now you have to use only the parameter **FORMAT** without the parameter **MAXRECL**. See examples at the end of this section.

VFTYP=X means, that a file will be automatically converted with `rvsut2fv` when sending a file. Example: VFTYP=X, FORMAT=F, MAXRECL=80. The difference to VFTYP=T is, that the lines do not need to have the desired length; it will be done by `rvsut2v`.

VFTYP=U means, that a binary (unstructured) file will be automatically converted with `rvsut2fv` when sending a file. Example: VFTYP=U, FORMAT=F, MAXRECL=80

VFTYP=D means, that files in fixed format are transmitted as binary files and files in format variable are transmitted as text files.

MAXRECL	(optional) maximal record length for the files in format **F** or **V**; this parameter should be used only if the files are not converted explicitly before with the tool `rvsut2fv`.
SIDORIG	Station ID of the virtual send station; the value of this parameter will be added to SFID for the transmission (see chapter 3.3 for SFID).

Repeatable Parameters:

Each group of repeatable parameters defines one receiver:

SID (required) station ID of receiver

UID (optional) user ID of receiver; if omitted or empty string: system at remote station

COMPRESSION (optional) if **Y** (Yes) the file should be compressed before sending

Example:

```
SEND /CREATE DSN=<file name>
(SID=<stationID> COMPRESSION=Y)
```

ENCRYPTION (optional) if **Y** (Yes) the file should be encrypted before sending

Example:

```
SEND /CREATE DSN=<file name> (SID=stationID
ENCRYPTION=Y)
```

DSNNEW (optional) virtual data set name used for transfer;

for transfer to MVS host, this must be a valid MVS data set name (acceptable to RACF)

default: virtual data set name is constructed from name of local data set

CODEOUT (optional) desired code of data set at receiver; (**A**=ASCII, **E**=EBCDIC)

Example:

```
SEND /CREATE DSN=<file name> FORMAT=U
CODEIN=a (SID=stationID CODEOUT=e)
```

CODETABLE (optional) defines the code table, which is to be used for the code conversion (see chapter about the code conversion in the User Manual)

Example:

```
SEND /CREATE DSN=/home/send/test22 FORMAT=V
CODEIN=A (SID=RFF CODEOUT=E
CODETABLE=/home/skk/arcdir/rtcaeown.dat)
```

`SEND /DELETE, /HOLD, /RELEASE` parameters:

DSN	(optional) name of local data set to be sent
SID	(optional) station ID of receiver
UID	(optional) user ID of receiver; defaults to empty string, i.e. the remote system
CMDID	(optional) unique command number of send request.

The send entry to be processed can be identified either by specifying **DSN**, **SID**, and **UID**, or by the unique command number **CMDID** of the send request.

If no parameter has been given or if more than one send request matches the specified parameters, `rvscal` does not modify any send request and returns with an error.

The following examples demonstrate, how commands may be specified in a data set that is used as input file to `rvsbat`, including use of continuation lines (the previous line ends with **+**) and comments (***** in column 1).

Examples:

```
*
*------------------------------------
*
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) +
    (SID=st2 UID=user2 CODEOUT=a TSTAMP=y
DSNNEW=dsnnew2)
*
*------ use defaults and serialize on LABEL
*
SEND    DSN=LPDBI.C +
SERIAL=y LABEL=l1 inittime=HOLD +
CODEIN=E FORMAT=U DISP=k +
+
     (SID=st1)
*
*------ serialize again on LABEL
*
SEND    DSN=rpu.c +
SERIAL=y LABEL=l1 inittime='1991/07/01 10:35' +
FORMAT=V +
+
     (SID=st1)
*
*------ serialize on data set (without specifying
the full data set name)
```

```
*
SEND    DSN=lpdbi.c +
SERIAL=y +
FORMAT=f +
+
     (SID=st1)
*
*------ serialize again on data set (and default
for FORMAT)
*
SEND    DSN=lpdbi.c +
SERIAL=y +
+
     (SID=st1)
*
*------ delete SEND request (unique)
*
SEND /D DSN=rpu.c (SID=st1)
*
*------- send file without converting with
rvsut2fv
*
SEND /C DSN=test.txt CODEIN=A FORMAT=F VFTYP=T
MAXRECL=80
(SID=RTT CODEOUT=E DSNNEW=FIX0GBE.TEXT)
*
*-------- send file converted with rvsut2fv
*
SEND /C DSN=test22.txt CODEIN=A FORMAT=V
(SID=RTT CODEOUT=E DSNNEW=FIX0GBE.TEXT)
*
*--------- example with almost all parameters
*
SEND /C DSN=newtest.rcl CODEIN=a DISP=k FORMAT=F
INITTIME='2003/01/02 22:10' SERIAL=y
LABEL=rechnung VFTYP=T MAXRECL=80 (SID=Z24
COMPRESSION=y ENCRYPTION=y DSNNEW=Z24R32
CODEOUT=e TIMESTAMP=y)
*
```

11.5. Command RESENTR

Function:

Create, update, or delete a resident receive entry.

Qualifiers:

`/CREATE`	(default) create resident receive entry
`/UPDATE`	update resident receive entry
`/DELETE`	delete resident receive entry

Parameters:

DSN	(required) virtual name of incoming data set
SID	(required) station ID of sender
ACCOUNT	(optional) accounting number or code
COMMENT	(optional) comment describing action of resident receive entry (up to 50 characters); default: empty string
DSNNEW	(optional) local name to be used for received data set; default: local data set name is constructed from virtual data set name
JOB	(optional) name of data set containing a job to be submitted after data has been received; if specified, the data set must exist. This batch file may contain substitution patterns. rvs® substitutes them before submitting the job to the operating system for execution:

- ?**DSN**?: name of local data set, where received information has been stored
- ?**VDSN**?: virtual data set name under which the data set was transmitted
- ?**DTAVAIL**?: date, when the data set was available for sending
- ?**FORMAT**?: record format of received data set:
  - **F** fixed record length
  - **V** variable record length
  - **T** text file (only ASCII characters)
  - **U** unstructured file (byte stream)
- ?**MAXRECL**?: The meaning of this field depends upon the record format of the received data set:
  - **F** format: length of each record
  - **V** format: maximum length a record may have
  - **T** and **U** format: always **0** (zero)
- ?**BYTES**?: number of transmitted bytes
- ?**RECORDS**?: number of transmitted records for **F** and **V** format data sets; always zero for **T** and **U** format data sets
- ?**DTRCV**?: date, when data set was delivered

to local user

- ?**LUID**?: recipient's (i.e. local) user ID
- ?**UID**?: sender's user ID
- ?**SID**?: sender's station ID
- ?**SIDDEST**?: Station ID of the virtual recipient's station
- ?**CNQS**? command number of EERP (End-to-End-Response) for received file
- ?**CNIE**? command number of the IE for received file
- ?**CNIZ**? command number of the IZ for received file
- ?**DSNTEMP**?: name of temp. data set (this data set should be deleted at the end of the job with the command:

– `DELETE ?DSNTEMP?` (**NT**)

– `rm ?DSNTEMP?` (**UNIX**)

– `DLTF ?DSNTEMP?` (**OS/400**)

default: no job will be submitted.

DISP (optional) determines what should be done with the incoming data set after processing has been completed: **D**=delete; **K**=keep

Important hint: If you use option **D**, the received file will be deleted and can not be used for further actions. It is only used as Trigger for the resident receive entry.

default: **K**

REPLACE (optional) determines action of rvs® when local data set does already exist and **DISP**=**K**;

may be **R** (replace existing data set), **N** (create new data set with unique name), or **I** (ignore incoming data set)

default: **N**

TSTAMP (optional) may be **Y**=yes or **N**=no; tells rvs, whether the data set name is to be timestamped to make it unique, when the data set is received;

default: **N**

CODETRANS It indicates whether the received file is to be converted to ASCII or EBCDIC, or an own code table is to be used (see chapter about the code conversion in the User Manual).

`e` code convertion from EBCDIC to ASCII

`a` code conversion from ASCII to EBCDIC

`t` code conversion with the own code table

Example:

```
resentr /c dsn=<received ASCII file>
codetrans=a sid=< Sender>
```

CODETABLE defines the code table, which is to be used for the code conversion (see chapter about the code conversion in the User Manual).

Example:

```
resentr /c dsn=<received EBCDIC file>
codetrans=t
codetable=/home/rvs/arcdir/rtcusrdat
sid=<Sender>
```

VFTYP Optional; you can specify here, if the received file is to be stored as a text file with a line feed after every record. This applies to files that are received in **Fixed** or **Variable** format only.

VFTYP=T the received file is to be stored as a text file with a line feed.

VFTYP=V the received file is to be stored in the rvs® intern format (without converting to text file).

VFTYP=S the received file is to be stored in the SINIX format.

Examples:

```
*
*---------- use all parameters
*
RESENTR /C DSN=incoming1    SID=st2 +
   DSNNEW=local.dsn REPLACE=n DISP=k +
   JOB=/home/rvs/bin/rcv.sh  COMMENT='This is a
test RE'
*
*---------- use defaults
*
RESENTR /C DSN=?DSN?   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
```

11.6. Command `SENDJOB`

Function:

Create, update, or delete a jobstart after send attempt entry

Qualifiers:

`/CREATE`	(default) create jobstart after send attempt entry
`/UPDATE`	update jobstart after send attempt entry
`/DELETE`	delete jobstart after send attempt entry

Parameters:

VDSN (required) virtual name of data set that is sent

SID (required) station ID of receiver

ATTEMPTS (optional) number of send attempts; determines in which case the job should start: value **0**: the job starts if data set has been transmitted successfully; value greater than **0**: determines the number of attempts to send the data set in vain after that the specified job should start

default: **0**

JOB (optional) name of data set containing a job to be submitted after data has been transmitted successfully or sending of data has been attempted in vain; if specified, the data set must exist

This batch file may contain substitution patterns. rvs® substitutes them before submitting the job to the operating system for execution:

- ?**DSN**?: name of local data set, that has been sent; In case of an **EERP** we don't have a local data set name. The value of ?**DSN**? has the appearance **QS** (**SIDORIG** - **SIDDEST**) with meaning:
  SIDORIG sender's station ID
  SIDDEST receiver's station ID

- ?**VDSN**?: virtual data set name under which the data set was transmitted
- **?DTAVAIL?:** date, when the data set was available for sending
- ?**FORMAT**?: Record format of the file sent
  - **F** fixed
  - **V** variable
  - **T** text
  - **U** unstructured
- ?**BYTES**?: Number of bytes transmitted
- ?**RECORDS**?: Number of records transmitted with **F** or **V** format
- ?**DTRCV**?: date, when data set was delivered to local user
- **?LABEL?:** string if the send command contained a **LABEL** parameter. Can be used to identify the send command.
- ?**SECN**?: command number of send command (**CN** of **SE**). Can be used to identify the send command
- ?**SKCN**?: Number of the send command
- ?**UID**?: sender's user ID; In case of **EERP** the value is always "!-**QS**-!"
- ?**SID**?: receiver's station ID
- ?**SIDORIG**?: StationID of the virtual send station
- ?**SENDATT**?: number of unsuccessful attempts after which the program is to be start
- **?DSNTEMP?**: name of temp. data set; this data set should be deleted at the end of the job with the command:

– `DELETE ?DSNTEMP?` (**NT**)

– `rm ?DSNTEMP?` (**UNIX**)

– `DLTF ?DSNTEMP?` (**OS/400**)

COMMENT	(optional) comment describing action of resident receive entry (up to 50 characters); default: empty string

Examples:

```
*
*---------- 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=?DSN?    SID=st2
JOB=/home/rvs/bin/sendok.sh
```

11.7. Command USER

Function:

- modify rvs® user table
- set user name
- set dialog language for user
- set privileges for user

Qualifiers:

`/CREATE`	create entry in rvs® user table
`/DELETE`	delete entry in rvs® user table
`/UPDATE`	update entry in rvs® user table

If no qualifier has been specified, the entry is either created or updated depending on whether it already exists or not.

Parameters:

UID	(optional) user ID which identifies the entry to be modified; a value different from the user ID of the current user may be specified only by a privileged user;

	default: current user
NAME	(optional) new name of user
LANGUAGE	(optional) new language setting (e.g. **E**=English, **D**=Deutsch)
PRIV	(optional) new privilege for user; (**U**=user, **O**=operator, **A**=administrator)

Examples:

```
*---------- 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
```

11.8. Command ACTIVATE

Function:

- Activate a partner station:
- rvs® will start a sender process, which establishes the connection to the partner station. All queued files will be transmitted. After that, the connection is closed.

Parameters:

SID	(required) station ID of partner

Examples:

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

11.9. Command `MODST`

Function:

Modify station table.

Attention: **This command overwrites old database entries**; other existing entries will not be deleted (only **DELST** will delete an entry)

Parameters:

DSN	(required) name of data set which contains a station table. the file name can be a single input file or a directory which contains several input files.

Examples:

```
*
*---------- 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"
```

11.10. Command `DELST`

Function:

Delete entry station table.

Parameters:

SID	Station ID of station to be deleted

Examples:

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

11.11. Command `LISTPARM`

Function:

- List the value of a rvs® parameter.
- In batch mode, the message "I: value" appears.
- In C programs, the value is given as a string in the `rvscal()` output parameter **S_MSG**.

Parameters:

parameter	name of rvs® parameter

Examples:

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

11.12. Command `SETPARM`

Function:

Set the value of a rvs® parameter:

Parameters:

parameter	name of rvs® parameter

Examples:

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

12. How to Work with rvs® C-CAL Interface

The following sections describes how to use the C-language functions that may be linked into an application program to execute rvs® utility commands. Please check `rvscal.h` that is contained in the rvs® distribution for last changes of structures and function prototypes.

Note: For all tasks (sending and receiving data files as well as administration of rvs®) you have to do with rvs® you can either

- call the general function `rvscal()` using string commands or
- use several dedicated functions

The string commands of `rvscal()` are identical to those useable for `rvsbat`. The command syntax is described in the chapter 11 "Description of Commands".

On the other hand the dedicated functions are on a lower abstraction level thus providing more control and flexibility. They are described in the following chapters.

12.1. Sending and Receiving with C-CAL Interface

This chapter describes the functions which are required to manage send entries using the C-CAL Interface. For all this functions a type definition and the corresponding prototypes are.

12.1.1. Type Definitions

```
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
defined 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
```

12.1.2. Get next send entry from Database

Prototype `rvsGetNextSend`:

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if we found a next send entry
	=**RVSCAL_END_FETCH**, if there are no send entries greater than **PREV_SEND_CN**
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured
prev_send_cn	(int, input)
	Command no. of previous send entry
info	(struct **INFO_SK ***, output)
	Struct with informations about next send entry with command number greater than **prev_send_cn**
REMARKS	`rvsGetNextSend` looks for the next send entry with a value greater than **prev_send_cn**. When the function is called the first time **prev_send_cn** must be **CN_START. CN_START** caused `rvsGetNextSend` to read all entries from the rvs® database and save it to an internal list. Every call of `rvsGetNextSend` with **prev_send_cn**=**CN_START** will refresh that list.

12.1.3. Get a send entry from Database

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if we found the informations about this send command
	=**RVSCAL_END_FETCH**, if there is no send entry with given command number
	=**RVSCAL_INTERNAL_ERROR**, if internal database error occured
send_cn	(int, input)
	Command number of a send entry
info	(struct **INFO_SK ***, output)
	Struct with informations about send entry

12.1.4. Set debug mode

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if debug mode is set
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured
MODE	(int, input)
	mode type

12.1.5. Change status of SE

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if no error occured
	=**RVSCAL_INTERNAL_ERROR**, if internal database error occured
cn_se	(long int, input)
	command no. of SE (returned by `CreateSendEntry`)
SetCmd	(int, input)
	Command type
	SET_HOLD : Hold this send entry
	SET_RELEASE : Release this send entry
	SET_DELETE : Delete this send entry
szSID	(char *, input)
	Station ID of receiver
s_msg	(char *, output)
	Error message text in case of error

12.1.6. Get next information entry

Prototype `rvsGetNextIE`:

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if no error occured.
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured
prev_ie_cn	(long int, input)
	command no. of previous info entry
p_info	(struct **INFO_SK** *, output)
	data struct containing info about entry

12.1.7. Send a File

Prototype `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);
```

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if it was successfull
	=**RVSCAL_INTERNAL_ERROR**, if error occured
dsn	(char *, input)
	filename of local file (details: see rvs® User Manual)
disp	(char *, input)
	disposition (**K**=keep, **D**=delete after transmission)
format	(char *, input)
	file format (**T**/**F**/**V**/**U**)
codein	(char *, input)

	code of input file (**A**=ASCII, **E**=EBCDIC)
inittime	(char *, input)
	time of earliest send attempt (**YY**/**MM**/**DD** **HH**:**MM**:**SS**)
serial	(char *, input)
	serialization flag (**Y** or **N**)
label	(char *, input)
	label of file (for serialization)
tstamp	(char *, input)
	timestamp
sid	(char *, input)
	station ID of receiver
dsnnew	(char *, input)
	virtual file name (**OFTP**)
codeout	(char *, input)
	output code (**A**=ASCII, **E**=EBCDIC)
s_msg	(char *, output)
	error message in case of error

12.1.8. Create a Send Entry

Prototype `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);
```

Description of Parameters

FUNCTIONVALUE	(int)
	=command number of send entry, if it was successfull
	=**RVSCAL_ERROR_CREATESEND**, if error

occured

PARAMETERS see chapter "Send a File"

12.2. Administration with C-CAL Interface

This chapter describes the functions to manage

- Station Table Entries
- rvs® Parameters
- rvs® Operator Commands
- Resident Receive Entries
- Entries for Jobstart after Send Attempt
- User Entries
- Database Functions

For all this functions a type definition is given as well as the corresponding prototypes and return codes.

12.2.1. Functions to manage Station Table Entries

This chapter describes the functions which are required to manage the Station Table entries.

Type Definitions

```
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;
```

Get next station entry from Database

Prototype `rvsGetNextStation`:

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

Description of Parameters

SIDPRE	(char *, input)
	SID of previous station
SID	(char *, output)
	SID of next station found in **ST** Table

Update station entry from Database

Prototype `rvsUpdateStation`:

```
PROCDEF int PROCKEYW rvsUpdateStation(
INFO_STATION *info);
```

Description of Parameters

info	(**INFO_STATION** *, input)
	Struct with informations about station entry

Get station entries from Database

Prototype `rvsGetStation`:

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

Description of Parameters

psz_SID	(char *, input)
	SID of station
info	(**INFO_STATION** *, output)
	Struct with informations about station entry

Delete station entry from Database

Prototype `rvsDeleteStation`:

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

Description of Parameters

psz_SID	(char *, input) SID of station

Free all suspended Commands

Prototype `rvsFreeSuspendedCommands`:

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

Description of Parameters

there is no parameter

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds.
	=**RVSCAL_END_FETCH**, if there are no stations with this **SID** or no entries greater than **SIDPRE**
	=**RVSCAL_DBERROR**, if database could not be opened
	=**RVSCAL_NEITHER_X_NOR_ISDN**, if neither X25 address nor ISDN address is specified
	=**RVSCAL_NEIGHBOR_STATION**, if there are routing links to this station in case of delete
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.2. Functions to manage rvs® Parameters

This chapter describes the functions which are required to manage rvs® parameters.

Type Definition

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

Get parameter value from Database

Prototype `rvsGetParm`:

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

Description of Parameters

parm	(char *, input)
	parameter name
stparm	(**PARM_STRUCT** *, output)
	Struct with informations about parameter entry

Get next parameter from Database

Prototype `rvsGetNextParm`:

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

Description of Parameters

parm	(char *, input)
	previous parameter
stparm	(**PARM_STRUCT** *, output)
	Struct with informations about the next parameter entry

Writes parameter value into Database

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

Description of Parameters

parm	(char *, input) parameter name
stparm	(**PARM_STRUCT** *, input) Struct with informations about parameter entry

Return Codes

FUNCTIONVALUE	(int) =**RVSCAL_OK**, if function succeeds =**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect =**RVSCAL_INVALID_NAME**, if parameter name doesn't exists =**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.3. Functions to manage rvs® Operator Commands

This chapter describes the functions which are required to manage rvs® Operator commands.

Store operator command into Database

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

Description of Parameters

command	(char *, input) command string

Wake the Monitor

Prototype `rvsWakeMonitor`:

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

Description of Parameters

there are no parameter

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_INVALID_NAME**, if parameter name doesn't exists
	=**RVSCAL_RC_WAKE_FAILED**, if wake command failes
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.4. Functions to manage Resident Receive Entries

This chapter describes the functions which are required to manage Resident Receive Entries.

Type Definition and Macros

```
    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
```

Get next command number of Resident Receive Entry from Database

Prototype `rvsGetNextRE`:

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

Description of Parameters

cn_pre	(const long, input) command number of previous resident receive entry
lpcn_re	(long *, output) command number of next resident receive entry found in **RE** table

Get Resident Receive Entry from Database

Prototype `rvsGetRE`:

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

Description of Parameters

cn_re	(const long, input) command number of resident receive entry
reinfo	(**INFO_RE** *, output) struct with informations about resident receive entry

Configure Resident Receive Entries

Prototype `rvsResidentResceiveEntry`:

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

Description of Parameters

icmd	(const int, input) command to specify what should be done **RE_UPDATE** - updates a resident receive entry **RE_DELETE** - deletes a resident receive entry **RE_CREATE** - creates a resident receive entry

reinfo	(INFO_RE *, input) struct with informations about resident receive entry

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_END_FETCH**, if there is no matching resident receive entry
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_INVALID_DSN**, if invalid **DSNNEW** was spezified
	=**RVSCAL_INVALID_JOB**, if invalid JOB was spezified
	=**RVSCAL_INVALID_NAME**, if parameter name doesn't exists
	=**RVSCAL_INVALID_SID**, if **SID_SENDER** isn't known
	=**RVSCAL_NOT_PRIVILEGED**, if user is not privileged in to configure resident receive entries
	=**RVSCAL_DBERROR**, if database could not be opened or closed
	=**RVSCAL_RE_NOT_FOUND**, if resident receive entry could not be found
	=**RVSCAL_DUPLICATE_RE**, if duplicate resident receive entry has found in case of **icmd**=**RE_CREATE**
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.5. Functions to manage Entries for Jobstart after Send Attempt

This chapter describes the functions which are required to manage Entries for Jobstart after Send Attempt.

Type Definition and Macros

```
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
```

Get next command number of Job Start Entry from Database

Prototype `rvsGetNextJS`:

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

Description of Parameters

cn_pre	(const long, input)
	command number of previous entry
lpcn_js	(long *, output)
	command number of next entry for jobstart after a send request found in **JS** table

Get Jobstart Entry from Database

Prototype `rvsGetJS`:

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

Description of Parameters

cn_js	(const long, input)
	command number of jobstart entry
jsinfo	(**INFO_JS** *, output)
	struct with informations about jobstart entry

Configure Entries for Jobstart after Send Attempt

Prototype `rvsJobStartEntry`:

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

Description of Parameters

icmd	(const int, input)
	command to specify what should be done **JS_UPDATE** - updates a jobstart entry

	JS_DELETE - deletes a jobstart entry **JS_CREATE** - creates a jobstart entry
jsinfo	(**INFO_JS** *, input) struct with informations about job start entry

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_END_FETCH**, if there is no matching jobstart entry
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_INVALID_DSN**, if invalid **DSNNEW** was spezified
	=**RVSCAL_INVALID_JOB**, if invalid **JOB** was spezified
	=**RVSCAL_INVALID_NAME**, if parameter name doesn't exists
	=**RVSCAL_INVALID_SID**, if **SID_RECEIVER** isn't known
	=**RVSCAL_NOT_PRIVILEGED**, if user is not privileged in to configure entries for Jobstart after Send Attempt
	=**RVSCAL_DBERROR**, if database could not be opened or closed
	=**RVSCAL_JS_NOT_FOUND**, if job start entry could not be found
	=**RVSCAL_DUPLICATE_JS**, if duplicate jobstart entry was found in case of **icmd** = **JS_CREATE**
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.6. Functions to manage User Entries

This chapter describes the functions which are required to manage User Entries.

Type Definition and Macros

```
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
```

Get next User from Database

Prototype `rvsGetNextUser`:

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

Description of Parameters

userpre	(char *, input)
	previous user name
user	(char *, output)
	next user name found in **BT** table

Get User Entry from Database

Prototype `rvsGetUser`:

```
PROCDEF int PROCKEYW  rvsGetUser( char *user,
INFO_USER *usinfo);
```

Description of Parameters

user	(char *, input)
	user name
usinfo	(**INFO_USER** *, output)
	struct with informations about user entry

Configure User Entries

Prototype `rvsUser`:

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

Description of Parameters

icmd	(const int, input)
	command to specify what should be done **USER_UPDATE** - updates a user **USER_DELETE** - deletes a user **USER_CREATE** - creates a user

usinfo	(**INFO_USER** *, output) struct with informations about user entry

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_END_FETCH**, if there is no matching user entry
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_INVALID_OWN_PRIV**, if user wants to decrease his own privileges
	=**RVSCAL_INVALID_UID**, if parameter **UID** is empty
	=**RVSCAL_INVALID_USER**, if user does'nt exist or wants to delete himself
	=**RVSCAL_NOT_PRIVILEGED**, if user is not privileged in to configure user entries
	=**RVSCAL_DBERROR**, if database could not be opened or closed
	=**RVSCAL_USER_NOT_FOUND**, if user entry could not be found
	=**RVSCAL_DUPLICATE_USER**, if duplicate user entry has found in case of **icmd** = **USER_CREATE**
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

12.2.7. rvs® Database Functions

This chapter describes the functions which are required to manage rvs® Database functions.

Type Definition and Macros

```
#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
```

Dump Database

Prototype `rvsDumpDB`:

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

Description of Parameters

environment	(char *, input) name of the environment data set, If environment is **NULL** or an empty zero terminated string rvs® will look for the default environment
dsn	(char *, input) data set name of dump file

Recover Database

Prototype `rvsWriteDB`:

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

Description of Parameters

environment	(char *, input) name of the environment data set, If environment is **NULL** or an empty zero terminated string rvs® will look for the default environment
dsn	(char *, input) data set name of dump file

Initialize Database

Prototype `rvsInitDB`:

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

Description of Parameters

environment	(char *, input) name of the environment data set, If

	environment is **NULL** or an empty zero terminated string rvs® will look for the default environment
sid_loc	(char *, input) local station ID

Delete Database

Prototype `rvsDeleteDB`:

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

Description of Parameters

environ-ment	(char *, input) name of the environment data set, If environment is **NULL** or an empty zero terminated string rvs® will look for the default environment
delattrib	(const int, input) attribute indicates which files should be deleted, possible values are **DEL_DB** (default) - remove all database files, **DEL_LOG** - remove *.logfiles, **DEL_TMP** - remove all files out of the temporary directory of rvs®

Dump User-, Receive, Js- and Station Entries

Prototype `rvsDumpRU`:

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

Description of Parameters

environ-ment	(char *, input) name of the environment data set, If environment is **NULL** or an empty zero terminated string rvs® will look for the default environment
dsn	(char *, input) data set name of dump file. The data set contents will get the input format useable with the rvs® batch interface.
dump-attrib	(const int, input) attribute indicates which files should be dumped,

possible values are

DUMP_USER (default) **-** dump all user entries,

DUMP_RES - dump all entries for Jobstart after Send request **DUMP_JS** - **DUMP_STATION**

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_DSN_NOT_EXIST**, if given data set does not exist
	=**RVSCAL_INTERNAL_ERROR**, if internal database error is occured

Get the version of rvs® Database

Prototype `rvsGetVersion`:
```
PROCDEF int PROCKEYW rvsGetDBVersion( char
*pszDBVersion);
```

Description of Parameters

pszDBVersion	(char *, output)
	version of rvs® database

Return Codes

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if function succeeds
	=**RVSCAL_PARAMETER_CHECK**, if at least one parameter is incorrect
	=**RVSCAL_ENVIRONMENT_NOT_EXIST**, if environment data set does not exist
	=**RVSCAL_ERROR_GETVERSION**, if version couldn't be determined

12.2.8. Other Functions

This chapter describes how to get SID from ODETTE and vice versa as well as how to get the list status of rvs® commands.

Get SID from ODETTE ID or vice versa

Prototype `rvsgetsid`:

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

Description of Parameters

FUNCTIONVALUE	(char *)
	=**NULL**, if no **SID** found or **DB** error.
	=pointer to string containing rvs® **SID**
s_odette_id	(char *, input)
	pointer to string containing ODETTE ID (max 26 bytes)

Prototype `rvsgetodid`:

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

Description of Parameters

FUNCTIONVALUE	(char *)
	=**NULL**, if no ODETTE ID found or **DB** error
	=pointer to max. 26 byte string
s_sid	(char *, input)
	pointer to string containing rvs® SID

List status of rvs® Commands

Prototype `rvslistcmd`:

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

Description of Parameters

FUNCTIONVALUE	(int)
	=**RVSCAL_OK**, if `rvslistcmd` succeeded
l_cmdid	(long, input)
	the **CMDID** (command number) of the command which was processed (return value of `rvscal`)
p_info	(struct **RVSCMD** *, output)
	pointer to a struct which contains information about the command **CMDID**
RVSCMD.status	(char, output)

	character which contains the status of the command: **a**/**d**/**e**/**h**/**p**/**q**/**s/f/...**)
RVSCMD.errorcode	(short, output)
	short integer which contains an error code, if it is different from **0**

Glossary

A

Access Method	The access method describes the way by which two stations are connected.
ASCII	American National Standard Code for Information Interchange

B

Batch Interface (`rvsbat`)	The batch interface of rvs® offers user functionality for automatic background use.

C

Communication Module (`rvscom`)	The communication module of the rvs® system connects to another station and sends or receives files.

D

Dialogue Interface (`rvsdia`)	The dialogue interface of rvs® provides interactiv user functionality.

E

EBCDIC	Extended Binary Coded Decimal Interchange Code
EDI	**E**lectronic **D**ata **I**nterchange
EDIFACT	**E**lectronic **D**ata **I**nterchange for **A**dministration **C**ommerce and **T**ransport
EERP	End-to-End-Response; ODETTE expression
ET	(`EmpfängerTabelle'; table of recipients) internal rvs® table describing one recipient
ETSI	European Telecommunications Standards Institute

F

G

H

HPFS	High Performance File System (OS/2)

I

IE	(`InformationsEingang'; information reception) internal rvs® command controlling delivery and routing of received files
IZ	(`InformationsZustellung'; information delivery) internal rvs® command delivering received files to one local recipient

J

K

L

M

MasterTransmitter (`rvsxmt`)	The MasterTransmitter of the rvs® system coordinates send and receive processes to ensure the optimal use of the net capacity.
Monitor (`rvsmon`)	The monitor is the main task of a rvs® system. It controls all other processes and initiates automatic follow up jobs if necessary.

N

O

ODETTE	Organization for Data Exchange by Tele

	Transmission in Europe
OFTP	ODETTE File Transfer Protocol The ODETTE File Transfer Protocol is the definition of a file transfer protocol by the ODETTE Group IV for OSI Layers 4 to 7. International Protocol used in many business fields (Industry, Commerce, Finance, ..).
Operator Console (`rvscns`)	The operator console provides the administrator with rvs® functions to control the rvs® system.
OSI	Open System Interconnection

P

PDF	Portable Document Format
Protocol	To connect two different computers they have to follow the same protocol. This protocol defines actions and reactions as well as the ”language” spoken.

Q

R

RE	(`Residenter Empfangseintrag'; resident receive entry) rvs® table describing actions to take, when a particular data set is delivered to a local recipient
`rvsmon`	See Monitor

S

SE	(`SendeEintrag'; send entry) internal rvs® command controlling sending of files
Send Entry	Order to rvs® which file has to be sent to which station. This entry is saved in the database.
SID	Station identification, rvs® internal name of an OFTP partner station

SK	(`SendeKommando'; send command) internal rvs® command controlling transfer of one file to one neighboring node
Station	A station is a node that can be addressed within a rvs® network. Each station is identified by a unique station ID (SID).

T

Transfer Component	Control program and line driver for a special access method

U

V

VDA	Verband der Deutschen Automobilhersteller Adress: Verband der Automobilindustrie e.V. (VDA) Abt. Logistik Postfach 17 05 63 60079 Frankfurt Tel.: 069-7570-0
VDSN	Virtual Data Set Name name under which file is known during transfer and for delivery

W

X

Y

Z

Index