Satellites International Limited

The Paddock
Hambridge Road
Newbury
Berkshire
United Kingdom

DBV-OSI II

Offene Kommunikation der Fachinformations- und Bibliothekssysteme

SOFTWARE

INTERNAL SPECIFICATION

468-SIS-004

Issue 1

D-022

December 1995

Prepared by:	J. Hough	SIL
	C. Attewell	SIL
	R. Bull	SIL
Approved by:	R. A. Bull	Project Manger
Authorised by:	A. K. Ward	Technical Director
Accepted by:		DDB

The content of this document is Commercial in Confidence and is the property of Satellites International Limited. The information contained herein must only be used for the purpose for which it is supplied and on request the document will be returned to Satellites International Limited together with any copies made.

DOCUMENT STATUS SHEET

DOCUMENT STATUS SHEET			
1. DOCUMENT TITLE: **DBV-OSI II Software Internal Specification 468-SIS-004 - D-022**			
2. ISSUE	3. REVISION	4. DATE	5. REASON FOR CHANGE
Prep.	1	29 Nov 94	Preparatory draft
Prep.	2	28 March 95	Preparatory draft rev. 2
Draft A	1	18 May 95	First Draft Issued to Partners
Issue 1	1	Dec. 1995	Raised to Issue

Table of Contents

1. INTRODUCTION 7
1.1 Purpose 7
1.2 Scope 7
1.3 Definitions, Acronyms and Abbreviations 8
1.4 References 8
1.5 Overview 8

2 PROJECT STANDARDS, CONVENTIONS AND PROCEDURES 10
2.1 Design Standards 10
2.2 Documentation Standards 10
2.2.1 Project Documents 10
2.2.2 Listings 10
2.2.3 File Header 10
2.3 Naming Conventions 12

3 PROGRAMMING STANDARDS 13

4. SOFTWARE DEVELOPMENT TOOLS 15
4.1 GNU C Compiler - Version 2.5.7 (and later versions) 15
4.2 GNU Make Utility - Version 3.70 15
4.3 GNU GDB Debugger - Version 4.11 16
4.4 Snacc Compiler - Version 1.1 16
4.5 X-Windows/OSF Motif Tool Kit 16

5 INTERNAL STATE MACHINE 17
5.1 Introduction 17
5.2 Reading the State Machine information 17
5.3 Origin State Machine 18
5.4 Target State Machine 21

6 DBV-OSI II API SPECIFICATION 24
6.1 Specification Layout Description 24

7 ORIGIN FUNCTIONS 26
7.1 Association Handling Functions Using ACSE-Services 26
7.1.1 DbvAssociateRequest() Function 27
7.1.2 DbvReleaseRequest() Function 30
7.1.3 DbvAbortRequest() Function 33
7.1.4 DbvConfirmAssociateRequest() Function 36
7.1.5 DbvConfirmReleaseRequest() Function 40
7.2 Association Handling Functions Using TCP-Services 43
7.2.1 DbvAssociateRequest() Function 43
7.2.2 DbvReleaseRequest() Function 46
7.2.3 DbvAbortRequest() Function 49
7.2.4 DbvConfirmAssociateRequest() Function 51
7.2.5 DbvConfirmReleaseRequest() Function 54

7.4 SR-/Z39.50 Functions 56
7.4.1 DbvInitializeRequest() Function 56
7.4.2 DbvSearchRequest() Function 61
7.4.3 DbvPresentRequest() Function 65
7.4.4 DbvDeleteResultSetRequest() Function 69
7.4.5 DbvScanRequest() Function 73
7.4.6 DbvResourceReportRequest() Function 77
7.4.7 DbvSortRequest() Function 81
7.4.8 DbvExtendedServicesRequest() Function 85
7.4.9 DbvCloseRequest() Function 89
7.4.9.1 DBVClose() Function - A Subordinate Of Functions DbvCloseRequest() And DbvCloseResponse() 91
7.4.10 DbvCloseResponse() Function 95
7.4.11 ReceiveDataOrigin() Function 97

8 TARGET FUNCTIONS 101
8.1 Association Handling Functions Using the ACSE-Services 101
8.1.1 DbvTargetInitialize() Function 101
8.1.2 DbvReceiveAssociateRequest() Function 104
8.1.3 DbvAssociateResponse() Function 107
8.1.4 DbvReleaseResponse() Function 110
8.1.5 DbvAbortRequest() Function 113
8.2 Association Handling Functions Using TCP Services 114
8.2.1 DbvTargetInitialize() Function 114
8.2.2 DbvReceiveAssociateRequest() Function 116
8.2.3 DbvAssociateResponse() Function 118
8.2.4 DbvReleaseResponse() Function 120
8.2.5 DbvAbortRequest() Function 122
8.4 SR-/Z39.50 Functions 123
8.4.1 DbvInitializeResponse() Function 123
8.4.2 DbvSearchResponse() Functions 127
8.4.3 DbvPresentResponse() Functions 131
8.4.4 DbvDeleteResultSetResponse() Functions 135
8.4.5 DbvScanResponse() Functions 139
8.4.6 DbvResourceReportResponse() Functions 143
8.4.7 DbvSortResponse() Functions 147
8.4.8 DbvSegmentationRequest() Functions 151
8.4.9 DbvExtendedServicesResponse() Functions 155
8.4.10 DbvCloseRequest() Functions 159
8.4.11 DbvCloseResponse() Functions 160
8.4.12 DbvReceiveDataTarget() Functions 161

9 UTILITY FUNCTIONS 165
9.1 SIGetEventLocation() Function 165
9.2 SISetACSEFile() Function 169
9.3 SIWhichStack() Function 171
9.4 SISetTraceParameter() Function 173
9.5 Structure and Type Definitions for Utilitiy Functions 176
9.5.1 DbvAssocId: 176
9.5.2 DbvSecs: 176
9.5.3 DbvFds: 176
9.5.4 DbvEventLocation: 176
9.5.5 DbvProtocolStack: 176

9.5.6 DbvTraceLevel: 177

10 INTERNAL FUNCTIONS 178
10.1 TCP/IP Association Control 178
10.1.1 DBVtcp_connect_to_host() Function 178
10.2 OSI Association Control 180
10.2.1 DBVOsiConnectToHost() Function 180
10.2.2 DBVOsiCompleteDbvAssociateResult() Function 183
10.3 ACSE Configuration File Functions 184
10.3.1 DBVGetACSEDetails() Function 184
10.3.2 DBVGetACN() Function 185
10.3.2 DBVReadACNS(), DBVConvertToOID() and DBVValidateACN() Functions 186
10.3.2 DBVGetPCDL(), DBVReadPCDLs() and DBVAddTheApplicationContexts() Functions 187
10.4 Specifying Target by Name Functions 188
10.4.1 DBVConvertNameToAddress() and DBVCompleteAddress() 188
10.5 State Machine Functions 189
10.5.1 DBVChangeState() Function 189
10.5.2 DBVOriginSMError(), DBVTargetSMError() Functions 189
10.6 Sending/Receiving Data 191
10.6.1 DBVencode_and_send() Function 191
10.6.3 DBVreceive_data_over_socket() Function 193
10.6.4 DBVreceive_data_over_OSI() Function 194
10.7 Trace File Control functions 196
10.7.1 DBVWrite() Function 196
10.7.2 DBVTraceLevel() Function 197
10.7.3 DBVTrace() Function 198
10.7.4 DBVErrorTrace() Function 199
10.7.5 DBVEnterFuncTrace() function 200
10.7.6 DBVLeaveFuncTrace() Function 201
10.8 Service-Provider System Log File Functions 202
10.8.1 DBVWriteOriginLog(), DBVWriteTargetLog(), DBVWriteLog(), DBVWriteErrorToLog(), DBVLogISODEACSEError() and DBVLogISODEPError() Functions 202
10.9 Miscellaneous Functions. 203
10.9.1 DBVfill_out_event_location_structure() Function 203
10.10 PDU Checking And Mapping Subordinate Functions 205

11 GLOBAL VARIABLE AND STRUCTURE INFORMATION 293
11.1 Origin 293
11.2 Target 293
11.3 Common 293

12 ERROR STRUCTURES, CODES AND EXPLANATIONS 295
12.1 Error Structures 295
12.2 DBV-OSI II Error codes: 296
12.2.1 General Error Codes: 296
12.2.2 Errors Associated with TCP/IP Connections: 297
12.2.3 Errors Associated with OSI Connections: 298
12.2.4 Errors associated with ADPU's: 298
12.2.5 State Machine Errors 299
2.2.6 Specifying Target by Name 300
12.2.7 Miscellaneous Errors 300
12.2.8 Memory Errors 301

12.2.9 ACSE Configuration File Errors..... 301

13 SPECIFYING TARGETS BY NAME..... 302
13.1 Introduction..... 302
13.2 The Name File Format..... 302
13.2.1 Example target_names.tbl File..... 303

14 THE ACSE CONFIGURATION FILE..... 304
14.1 Introduction..... 304
14.2 ACSE Configuration File Format..... 304
14.3 Example file..... 305
14.4 Further Information..... 305

15. REGISTERING AN OSI APPLICATION USING ISODE..... 306
15.1 Introduction..... 306
15.3 Using iaed..... 306
15.3.1 Registering An Application..... 306
15.3.2 Starting iaed..... 308
15.3 Using tsapd..... 308
15.3.1 isoservices Examples..... 308

16 CONFIGURING AN INTERNET DAEMON..... 310
16.1 Using targetd..... 310
16.1 Using inetd..... 310

Annex A: Data Definitions..... 311
A.1 Constants and Type Definitions..... 311
A.2 Enumeration Types..... 311
A.3 Structure/Union Types..... 314
A.3.1 Structures associated with Association Handling functions..... 314
A.3.2 Structures associated with SR-/Z39.50 functions..... 315
A.3.3 Structures common to all functions..... 316

Annex B: SR-/Z39.50 Data Structures..... 317
B.1 Global type and structure definitions..... 317
B.2 APDU Datastructures..... 319
B.3 APDU Substructures..... 325
B.4 Object Identifier, Record Syntaxes, Resource Report, Update Extended Service.... 336

1. INTRODUCTION

1.1 Purpose

This document describes the recommended approach to be used by each partner in the DBV-OSI II consortium for developing DBV-OSI II origin and target systems. The service-user/service-provider interface functions, the API functions, are described in chapters 7, 8, and 9. It is assumed that the reader is familiar with the DBV-OSI II project and the series of project documents.

1.2 Scope

DBV-OSI II is a software system which provides facilities allowing the development of Z39.50 version 3 based applications on TCP/IP and/or OSI networks.

The software system is a collection of UNIX libraries.

The DBV-OSI II libraries implement the service-provider component of the ANSI Z39.50 information search and retrieval protocol. Service-user applications are written by calling the library functions.

The software has been ported to the following computer systems:

- Sun Sparc2 running Solaris 1,
- Sun Sparc 10 running Solaris 2,
- Siemens-Nixdorf SINIX operating system,
- Hewlett Packard 9000/700 running HP_UX.
- 486 PC running Linux (TCP/IP only).

DBV-OSI II supports communication over TCP/IP and/or OSI networks provided the UNIX systems on which DBV-OSI II origin/targets run are correctly configured.

Most UNIX systems provide TCP/IP support as part of their environments.

To configure DBV-OSI II for use on OSI networks, an additional software package called ISODE must be installed on the system.

An OSF-Motif based administration tool is provided which allows visibility and control of target sessions when the system is active.

Note, there are many references in this document that refer to ...z3950_1994.... This is because at the time of initial writing, the protocol ASN.1 specification from the pre-final release of the Z39.50 document was used. The final release of the document had this changed to 1995. This does not affect the support for the protocol by this package in any way.

1.3 Definitions, Acronyms and Abbreviations

The following lists the definitions and abbreviations used in this document. The notation and syntax of the diagrams used in this document is given in Appendix A.

API	Application Program Interface
ANSI	American National Standards Institute
APDU	Application Protocol Data Unit
FIFO	First-In, First-Out UNIX inter-process commuication mechanism
GUI	Graphical User Interface
IR	Information Retrieval
OSF	Open Software Foundation
SW	Software
SIL	Satellites International Limited
TBD	To be defined
TCP/IP	Transport Control Protocol / Internet Protocol
ISO	International Standards Organisation
BER	Basic Encoding Rules
Snacc	Sample Neufeld ASN.1 to C/C++ Compiler
ISODE	International Standards Organisation Development Environment

1.4 References

DBV-OSI II D-0xx documents.

ESA PSS-05-05, Guide to Detailed Design Document. Available from ESA Publications Division, ESTEC, Noordwijk, The Netherlands.

Snacc1.1: A High Performance ASN.1 to C/C++ Compiler

1.5 Overview

The depth of the architectural design in the SES was to a level where implementation independence was maintained. This document continues with the design implemented on the UNIX operating system, using an ANSI C compiler and utilisation of UNIX System V operating system services.

For each object, the following attributes will be described:

Identifier:	the naming identification of the object,
Type	the characteristics of the object,

Purpose	the traceability to respective software requirements,
Functionality	what the object does,
Subordinates	immediate child objects to this object,
Dependencies	object pre/post conditions and exclusive operations,
Interfaces	detailed description of object interaction to other objects
Resources	object requirements of its environment,
References	implicit references to other related documents,
Processing	description of object processing,
Data	description of object internal data usage,
Usage	an example of using the function.

2 PROJECT STANDARDS, CONVENTIONS AND PROCEDURES

2.1 Design Standards

The software consists of ANSI C source code and associated header files which are compiled then archived to form the DBV-OSI II libraries. Some of the C files within DBV-OSI II which contain Z39.50 encoding and decoding routines were automatically generated from a specially prepared ascii version of the Z39.50 version 3 specification using an 'ASN.1 compiler' called snacc 1.1.

2.2 Documentation Standards

2.2.1 Project Documents

This SIS document was produced using the PC windows wordprocessor 'Microsoft Word for Windows' (version 2).

2.2.2 Listings

All pages of code listing have a standard top banner containing the module filename, date of printing, page number, author and company.

2.2.3 File Header

A standard header exists at the top of all ASCII files used within the project including:

- C source code files
- C source code header files
- Compilation and script files
- Data/control files

The header contains

- name of the module and version number

- name of the programmer and date of completion

- date accepted for incorporation by the Quality Assurance manager

- a brief description of the module

in the following format:

```
/* Module Name:
          Version:
          Author:
          Satellites International Ltd
          Unit 3, The Paddock,
          Hambridge Road,
          Newbury,
          Berkshire, RG14 5TQ
          United Kingdom
          Date of Issue:
          QA Acceptance:
          Date of Acceptance:
          Description:
     */
```

Note that the Version and Author fields are entered 'automatically' by use of source code control system tokens which are automatically expanded.

2.3 Naming Conventions

The following naming conventions are used for all files in the project.

File Type	Name	Extension
document	<<mnemonic>>	.doc
C source file	IDENTIFIER	.c
C header file	IDENTIFIER	.h
Object File	IDENTIFIER	.o
Program make file	Makefile	no extension required
Executable file	IDENTIFIER	no extension required
TBL file	IDENTIFIER	.tbl

3 PROGRAMMING STANDARDS

All source code is appropriately formatted and commented. The ANSI C standard is used for all C source code.

All "include", "external" and "define" statements are made before the function declarations in the module.

Where appropriate, modules have an accompanying '.h' header file which contains prototype declarations of all the functions in the module together with relevant "include" and other statements.

Specific naming conventions apply to certain functions. There are two categories of functions

a) Those which may be called by the service-user

b) Those developed by SIL to implement the service user functions

Category a) - Service user functions

All API functions which may be called by the service-user (with the exception of Utility functions) are prefixed with:

Dbv

All Utility Functions which may be called by the service-user are prefixed with:

SI

Category b) - Service provider functions

Most internal functions developed by SIL are prefixed with

DBV

Some internal functions are derived from SIL library modules and use a different naming convention - they are prefixed by upper case letters where the module is accessible from other modules, or by lower case letters for functions internal to the module.

Internal global variables are prefixed with:

DBV

Conditional compiler pre-processor directives (eg, #defines) that are internal to the API are prefixed with:

DBV_

No internal functions or structures should never be called or used by the service user.

The type, name and descriptions of all variables/varaible groups used in each function start on a separate line.

Each function developed has the following format characteristics:

Function Type: All functions declare the type they return (e.g. void, int, float, double or character).

Function parameters: All variables input and output from a function are passed in the function declaration statement. Where appropriate the input variables shall appear first in a function calling statement.

Numerical Constants: Constants (e.g. 3.14159, 2.71828) do not appear in the code but are defined in header files. This makes the code easier to read and maintain.

Comments: Comments are written in normal English text using both upper and lower case characters.

Code Structure: Code is written in a structured form. GOTO statements and use of labels shall be kept to a minimum.

Indentation: Blocks of code contained within loops and IF, ELSE IF structures are indented by 8 spaces for each level of nesting. When code modules are completed they are parsed with a UNIX C-beautifier utility.

Logical Operators: Any conditional evaluation which uses more than one logical operation uses parentheses to emphasise the order of evaluation. Complicated tests are broken down into separate components.

Mixed Types: When expressions contain mixed types explicit type coercion is used where possible.

4. SOFTWARE DEVELOPMENT TOOLS

The following software development tools are used for DBV-OSI II API development.

4.1 GNU C Compiler - Version 2.5.7 (and later versions)

All DBV-OSI II code was developed using the GNU C Compiler version 2.5.7 (or a later version), a freeware ANSI C/C++ compiler developed by the Free Software Foundation in America. The GNU C compiler is the compiler of choice for many UNIX developers as it is well supported, has regular free updates and is available for many platforms which aids multiple machine software porting.

It is recommended that all origin and target programs developed using DBV-OSI II are compiled using the GNU C compiler, version 2.5.7 or higher.

4.2 GNU Make Utility - Version 3.70

All DBV-OSI II software development builds were controlled by the GNU 'make' utility version 3.70, a freeware utility developed by the Free Software Foundadtion in America. This utility is compatible with all major proprietrary make utilities (such as Sun MicroSystem's make) and has additional useful features.

The purpose of a make utility is to determine automatically which elements of a large program need to be recompiled, and issue the commands to recompile them.

Make can be used with any programming language whose compiler can be run with a shell command. Make is not limited to programs, it can be used to describe any task where some files must be updated automatically from others whenever the others change.

To use a make utility, a "makefile" must be written which describes the relationships among files in a program, and states the commands for updating each file.

It is recommended that builds of all origin and target programs developed using DBV-OSI II use the GNU make utility, version 3.70 or higher.

4.3 GNU GDB Debugger - Version 4.11

All DBV-OSI II software development debugging were undertaken using the GNU 'gdb' symbolic debugger, version 4.11.

The purpose of a debugger such as GDB is to allow a program developer to see what is going on ``inside'' another program while it executes or what the program was doing at the moment it crashed.

GDB has four main functions with additional supporting functions to assist the developer in the debugging of a program:

- Start the program, specifying anything that might affect its behaviour.

- Make the program stop on specified conditions.

- Examine what has happened, when the program has stopped.

- Change things in the program, so that corrective action can be experimented with in order to learn about any other program bugs.

4.4 Snacc Compiler - Version 1.1

A utility called snacc 1.1 was used to generated various C files in DBV-OSI II which contain code to encode/decode C structures to/from 'BER' data, the low level data protocol of Z39.50.

To create these files, a specially prepared acsii version of the Z39.50 version 3 specification (specified using the ASN.1 language) was used as input to the utility. Once generated, the output files were hand edited in to cater for special encoding/decoding cases not handled by snacc.

Only the final versions of the C files are provided in DBV-OSI II.

4.5 X-Windows/OSF Motif Tool Kit

Suitable OSF-Motif development packages are used for the adminstrartion tool.

5 INTERNAL STATE MACHINE

5.1 Introduction

This section of the SIS details the Origin State Machine (OSM) and the Target State Machine (TSM). Each state table shows the incoming events to the protocol, the state changes which result and the state of the association being used for protocol communication.

The DBV-OSI II API state machines verify any actions taken by either the API user or the protocol information received from the network. When the state machine detects a violation by either party, the API user is notified via the `DbvError` structure (See Section 11.0 for details of all state machine errors) of the current function call.

In theory, a state machine violation from the incoming network data should never occur (and will not be possible between two DBV-OSI II based applications).

As well as checking for the correct usage of the Z39.50 protocol, attempts to improperly release the association will also cause state machine violations.

It should be noted that the internal state machines for both the origin and target are not implementations of the full Z39.50 v3 state machines. This is because Concurrent Operations, Access and Resource Control and some Extended Services are not implemented in the API.

It should also be noted that in the context of TCP, associations cannot be rejected by the API so, as long the address is valid, a DBV_OE_AASSOCIATERESP_IND_FALSE will never occur.

Under SR Version 1 it is not possible for the Target to initiate a negotiated release. If the origin receives a Release Request from the target, it will be reported to the service user as a DBV_OE_AABORT_IND event and the association will be aborted.

5.2 Reading the State Machine information

Each state diagram consists of three main sections: states, transitions and events. The states are listed across the top of the state table, while the incoming events are listed down the left hand side. The grid itself shows the transition to be taken (which state to move to). A blank grid box indicates that the incoming event is invalid for the particular state. An incoming event can be from either the network or the API user and will either cause a new state to be the current state or produce an error to the API user.

Any event ending in _PDU, or _IND was received from the network, while the rest are from the API user.

A blank box signifies that an error has occurred and is reported to the service user, but the current state has not changed.

Each transition has an implied action which is completed before the move to the next state. These actions are not listed, but can always be determined by the name of the event. For example, if the current origin state is 5 - DBV_OS_OPEN and the incoming event is DBV_OE_SEARCHREQ (caused by the API user calling the SearchRequest() API function), before

moving to state 4 - DBV_OS_SEARCH_SENT the API will send a SearchRequest PDU to the target.

5.3 Origin State Machine

Origin States:

No.	Name:	Description:
1	DBV_OS_CLOSED	The initial state. There is no association.
2	DBV_OS_ASSOCREQ_INPROGRESS	The origin has requested an association. The establishment of an association is in progress.
3	DBV_OS_ASSOCIATION	A valid association exists. The origin is awaiting an InitializeRequest from the API user.
4	DBV_OS_INITSENT	The origin has sent an InitializeRequest PDU to the Target
5	DBV_OS_OPEN	The origin is waiting for a SearchRequest, PresentRequest, DeleteResultSetRequest, ScanRequest, ResourceRequest, SortRequest, ExtendedServciesRequest or CloseRequest from the API user. An AbortRequest or ReleaseRequest is also valid. A CloseRequest PDU or AbortRequest PDU may also be received from the target at this moment.
6	DBV_OS_SEARCH_SENT	The origin has sent a SearchRequest PDU to the target
7	DBV_OS_PRESENT_SENT	The origin has sent a PresentRequest PDU to the target
8	DBV_OS_DELETE_SENT	The origin has sent a DeleteResultSetRequest PDU to the target
9	DBV_OS_SCAN_SENT	The origin has sent a ScanRequest PDU to the target
10	DBV_OS_RESOURCE_REPORT_SENT	The origin has sent a ResourceReportRequest PDU to the target
11	DBV_OS_SORT_REQUEST_SENT	The origin has sent a SortRequest PDU to the target
12	DBV_OS_EXTENDED_SERVICE_SENT	The origin has sent an ExtendedServicesRequest PDU to the target
13	DBV_OS_RELEASEREQ_SENT	The origin has sent an Association ReleaseRequest to the target
14	DBV_OS_CLOSE_SENT	The origin has sent a CloseRequest PDU to the target
15	DBV_OS_CLOSE_RCVD	The origin has received a CloseRequest PDU from the target
16	DBV_OS_ERROR	The origin has received an out of sequence PDU/failed to decode a PDU
17	DBV_OS_DEAD	The origin has been disconnected from the target

Incoming Origin Events

Incoming Event:	Description:
DBV_OE_IASSOCREQ	AssociateRequest called
DBV_OE_AASSOCIATERESP_IND_TRUE	AssociateResponse indication received. Association accepted (Received by AssociateRequest/ ConfirmAssociateRequest)
DBV_OE_AASSOCIATERESP_IND_FALSE	AssociateResponse indication received. Association rejected (Received by AssociateRequest/ ConfirmAssociateRequest)

DBV_OE_INITREQ	InitializeRequest sent
DBV_OE_INITRESP_PDU_TRUE	InitializeResponse PDU Received (Z39.50 session accepted)
DBV_OE_INITRESP_PDU_FALSE	InitializeResponse PDU Received (Z39.50 session rejected)
DBV_OE_SEARCHREQ	Send SearchRequest
DBV_OE_SEARCHRESP_PDU	SearchResponse PDU received
DBV_OE_PRESENTREQ	Send PresentRequest
DBV_OE_PRESENTRESP_PDU	PresentResponse PDU received
DBV_OE_DELETEREQ	Send DeleteResultSetRequest
DBV_OE_DELETERESP_PDU	DeleteResultSetResponse PDU received
DBV_OE_SCANREQ	Send ScanRequest
DBV_OE_SCANRESP_PDU	ScanResponse PDU received
DBV_OE_RESOURCEREQ	Send ResourceReportRequest
DBV_OE_RESOURCERESP_PDU	ResourceReportResponse PDU received
DBV_OE_SORTREQ	Send SortRequest
DBV_OE_SORTRESP_PDU	SortResponse PDU received
DBV_OE_SEGMENTREQ_PDU	SegmentationRequest PDU received
DBV_OE_EXTENDEDREQ	Send ExtendedServicesRequest
DBV_OE_EXTENDEDRESP_PDU	ExtendedServicesResponse PDU received
DBV_OE_IRELREQ	ReleaseRequest called
DBV_OE_ARELREQ_IND	Association ReleaseRequest Indication received
DBV_OE_ARELRESP_IND	Association ReleaseResponse Indication received
DBV_OE_AABORT_IND	Association Abort Indication
DBV_OE_IABORTREQ	AbortRequest called
DBV_OE_CLOSEREQ_PDU	CloseRequest PDU received
DBV_OE_CLOSEREQ	Send Close Request
DBV_OE_CLOSERESP	Send CloseResponse
DBV_OE_DECODE_FAILED	An attempt to decode a PDU failed

State Tables for Origin:

State / Event	DBV_OS_CLOSED 1	DBV_OS_ASSOCREQ_ INPROGRESS 2
DBV_OE_IASSOCREQ	2	
DBV_OE_AASSOCIATE-RESP_IND_TRUE		3
DBV_OE_AASSOCIATER ESP_IND_FALSE		17
. . .		
DBV_OE_AABORT_IND		17
DBV_OE_IABORT_REQ		17

State / Event	DBV_OS_ ASSOCIA-TION 3	DBV_OS_ INITSENT 4	DBV_OS_ OPEN 5	DBV_OS_ SEARCH_ SENT 6	DBV_OS_ PRESENT _SENT 7	DBV_OS_ DELETE_ SENT 8	DBV_OS_ SCAN_S-ENT 9	DBV_OS_ RESOUR-CE_REP-ORT_ SENT 10	DBV_OS_ SORT_ REQUEST _SENT 11	DBV_OS_ EXTEND-ED_SERV ICE_SENT 12	DBV_OS_ RELEAS-EREQ_ SENT 13	DBV_OS_ CLOSE_ SENT 14	DBV_OS_ CLOSE_ RCVD 15	DBV_OS ERROR 16	DBV_OS_ DEAD 17
DBV_OE_ INITREQ	4														
DBV_OE_ INITRESP _PDU_ TRUE	16	5	16	16	16	16	16	16	16	16	16	16	16		
DBV_OE_ INITRESP _PDU_ FALSE	16	3	16	16	16	16	16	16	16	16	16	16	16		
DBV_OE_ SEARCHREQ			6												
DBV_OE_ SEARCHRESP _PDU	16	16	16	5	16	16	16	16	16	16	13	14	16		
DBV_OE_ PRESENTREQ			7												
DBV_OE_ PRESENTRESP _PDU	16	16	16	16	5	16	16	16	16	16	13	14	16		
DBV_OE_ DELETEREQ			8												
DBV_OE_ DELETERESP _PDU	16	16	16	16	16	5	16	16	16	16	13	14	16		
DBV_OE_SCAN REQ			9												
DBV_OE_ SCANRESP _PDU	16	16	16	16	16	16	5	16	16	16	13	14	16		
DBV_OE_ RESOURCEREQ			10												
DBV_OE_ RESOURCERE-SP_PDU	16	16	16	16	16	16	16	5	16	16	13	14	16		
DBV_OE_ SORTREQ			11												
DBV_OE_SOR-TRESP_PDU	16	16	16	16	16	16	16	16	5	16	13	14	16		
DBV_OE_SEG-MENTREQ_ PDU	16	16	16	16	7	16	16	16	16	16	13	14	16		
DBV_OE_EXT-ENDEDREQ			12												
DBV_OE_ EXTENDEDR-ESP_PDU	16	16	16	16	16	16	16	16	16	5	13	14	16		
DBV_OE_ IRELREQ	13		13												
DBV_OE_AREL REQ_IND	16	16	16	16	16	16	16	16	16	16	16	16	16		
DBV_OE_AREL RESP_IND	16	16	16	16	16	16	16	16	16	16	17	16	16		
DBV_OE_ AABORT_IND	17	17	17	17	17	17	17	17	17	17	17	17	17	17	
DBV_OE_ IABORTREQ	17	17	17	17	17	17	17	17	17	17		17	17	17	
DBV_OE_ CLOSEREQ _PDU	16	16	15	15	15	15	15	15	15	15	13	3		15	
DBV_OE_ CLOSEREQ			14	14	14	14	14	14	14	14				14	
DBV_OE_ CLOSERESP													3		
DBV_OE_ DECODE_ FAILED	16	16	16	16	16	16	16	16	16	16		16	16		

5.4 Target State Machine

Target States:

No.	Name:	Description:
1	DBV_TS_CLOSED	The initial state. The Target is waiting for an Associate Request.
2	DBV_TS_ASSOC_INPROGRESS	
3	DBV_TS_ASSOCIATION	There is a valid association. The target is awaiting an InitializeRequest PDU
4	DBV_TS_INIT_RECVD	The target has received an InitializeRequest PDU and indicated this to the API user
5	DBV_TS_OPEN	The target is waiting for a SearchRequest, PresentRequest, DeleteResultSetRequest, ScanRequest, ResourceReportRequest, SortRequest, ExtendedServicesRequest or CloseRequest PDU from the origin. An AbortRequest or ReleaseRequest is also valid. A CloseRequest or Abort may also be recevied by the API user.
6	DBV_TS_SEARCH_RECVD	The target has received a SearchRequest PDU and indicated this to the API user.
7	DBV_TS_PRESENT_RECVD	The target has received a PresentRequest PDU and indicated this to ther API user.
8	DBV_TS_DELETE_RECVD	The target has received a DeleteResultSetRequest PDU and indicated this to the API user.
9	DBV_TS_SCAN_RECVD	The target has received a ScanRequest PDU and indicated this to the API user.
10	DBV_TS_RESOURCE_REPORT_RECVD	The target has received a ResourceReportRequest PDU and indicated this to the API user.
11	DBV_TS_SORT_RECVD	The target has received a SortRequest PDU and indicated this to the API user.
12	DBV_TS_EXTENDED_SERVICE_RECVD	The target has received a ExtendedServiceRequest PDU and indicated this to the API user.
13	DBV_TS_ARELEASE_REQ_RECVD	The target has received a ReleaseRequest
14	DBV_TS_CLOSE_SENT	The target has sent a CloseRequest PDU to the origin
15	DBV_TS_CLOSE_RECVD	The target has received a CloseRequest PDU from the origin
16	DBV_TS_ERROR	The target has received an out of sequence PDU/failed to decode a PDU
17	DBV_TS_DEAD	The target has been disconnected from the origin

Incoming Target Events:

Incoming Event:	Description:
DBV_TE_AASOCREQ_IND	Associate Request indication received.
DBV_TE_IASSOCRESP_TRUE	Associate Response (Accepted)
DBV_TE_IASSOCRESP_FALSE	Associate Response (Rejected)
DBV_TE_INITREQ_PDU	InitializeRequest PDU received
DBV_TE_INITRESP_TRUE	InitislizeResponse sent (Z39.50 session accepted)

DBV_TE_INITRESP_FALSE	InitializeResponse sent (Z39.50 session rejected)
DBV_TE_SEARCHREQ_PDU	SearchRequest PDU received
DBV_TE_SEARCHRESP	Send SearchResponse
DBV_TE_PRESENTREQ_PDU	Present Request PDU received
DBV_TE_PRESENTRESP	Send PresentResponse
DBV_TE_DELETEREQ_PDU	DeleteResultSetRequest PDU received
DBV_TE_DELETERESP	Send DeleteResultSetResponse
DBV_TE_SCANREQ_PDU	ScanRequest received
DBV_TE_SCANRESP	Send ScanResponse
DBV_TE_RESOURCEREQ_PDU	ResourceReportRequest PDU received
DBV_TE_RESOURCERESP	Send ResourceReportResponse
DBV_TE_SORTREQ_PDU	SortRequest PDU received
DBV_TE_SORTRESP	Send SortResponse
DBV_TE_SEGMENTREQ	Send SegmentationRequest
DBV_TE_EXTENDEDREQ_PDU	ExtendedServicesRequest PDU received
DBV_TE_EXTENDEDRESP	Send ExtendedServicesResponse
DBV_TE_ARELIND	Association ReleaseRequest Indication
DBV_TE_IRELRESP	ReleaseResponse called
DBV_TE_AABORTIND	Association Abort indication
DBV_TE_IABORTREQ	Call AbortRequest
DBV_TE_CLOSEREQ_PDU	CloseRequest PDU received
DBV_TE_CLOSEREQ	Send CloseRequest
DBV_TE_CLOSERESP	Send CloseResponse
DBV_TE_DECODE_FAILED	The decoding of an incoming PDU has failed

State tables for Target:

State / Event	DBV_TS_CLOSED 1	DBV_TS_ASSOC_INPROGRESS 2
DBV_TE_AASOCREQ_IND	2	
DBV_TE_IASSOCRESP_TRUE		3
DBV_TE_IASSOCRESP_FALSE		17
. . .		
DBV_TE_AABORT_IND		17
DBV_TE_IABORT_REQ		17

State / Event	DBV_TS_ASSOCIA-TION 3	DBV_TS_INIT_RECVD 4	DBV_TS_OPEN 5	DBV_TS_SEARCH_RECVD 6	DBV_TS_PRESENT_RECVD 7	DBV_TS_DELETE_RECVD 8	DBV_OS_SCAN_RECVD 9	DBV_TS_RESOUR-CE_REP-ORT_RECVD 10	DBV_TS_SORT_RECVD 11	DBV_TS_EXTEND-ED_SERVICE_RECVD 12	DBV_TS_ARELEAS-EREQ_RECVD 13	DBV_TS_CLOSE_SENT 14	DBV_TS_CLOSE_RCVD 15	DBV_TS_ERROR 16	DBV_TS_DEAD 17
DBV_TE_INITREQ_PDU	4	16	16	16	16	16	16	16	16	16			16		
DBV_TE_INITRESP_TRUE		5													
DBV_TE_INITRESP_FdALSE		3													
DBV_TE_SEARCHREQ_PDU	16	16	6	16	16	16	16	16	16	16		14	16		
DBV_TE_SEARCHRESP				5											
DBV_TE_PRESENTREQ_PDU	16	16	7	16	16	16	16	16	16	16		14	16		
DBV_TE_PRESENTRESP					5										
DBV_TE_DELETEREQ_PDU	16	16	8	16	16	16	16	16	16	16		14	16		
DBV_TE_DELETERESP						5									
DBV_TE_SCANREQ_PDU	16	16	9	16	16	16	16	16	16	16		14	16		
DBV_TE_SCAN RESP							5								
DBV_TE_RESOURCEREQ_PDU	16	16	10	16	16	16	16	16	16	16		14	16		
DBV_TE_RESOURCE-RESP								5							
DBV_TE_SORT REQ_PDU	16	16	11	16	16	16	16	16	16	16		14	16		
DBV_TE_SORT RESP									5						
DBV_TE_SEGMENTREQ					7										
DBV_TE_EXTENDEDREQ_PDU	16	16	12	16	16	16	16	16	16	16		14	16		
DBV_TE_EXTENDED-RESP										5					
DBV_TE_AREL_IND	13	16	13	16	16	16	16	16	16	16	16	16	16		
DBV_TE_IRELRESP											14				
DBV_TE_AABORT_IND	17	17	17	17	17	17	17	17	17	17		17	17	17	
DBV_TE_IABORTREQ	17	17	17	17	17	17	17	17	17	17		17	17	17	
DBV_TE_CLOSEREQ_PDU	16	16	15	15	15	15	15	15	15	15		3		15	
DBV_TE_CLOSEREQ			14	14	14	14	14	14	14	14				14	
DBV_TE_CLOSERESP													3		
DBV_TE_DECODE_FAILED	16	16	16	16	16	16	16	16	16	16		16	16		

6 DBV-OSI II API SPECIFICATION

6.1 Specification Layout Description

Each of the DBV-OSI II functions explained below follow a common format. The format is split into sections. Each section is explained below:

Identification:

This section describes the format for calling the function.

Type:

Lists the location of the function in its source and object form.

Purpose:

Provides an overview of the defined purpose of the function.

Functionality:

Describes the steps the function goes through to achieve its purpose and the states and meanings of any function parameters which have been altered. The return value, if applicable, is also explained.

Subordinates:

Lists other specific functions (not including system functions) which are called by the function being described.

Dependencies:

Lists any specific dependencies the function requires for it's successful completion. This includes software and hardware.

Interfaces:

Describes the use of and change to function parameters and data by the function.

Resources:

Any software resources used by the function (libraries, etc.).

References:

A list of any other important or related information about this function.

Processing:

Lists any important or useful logic contained within the function. This may take the form of pseudo-code representation.

Data:

Provides an explanation of any specific variable used internally by the function.

Usage:

Provides a code example of the usage of the function.

7 ORIGIN FUNCTIONS

NOTE: While the Association functions are split into two distinct sections in this document - OSI and TCP services - the actual code is integrated into the same source file and functions. There is logic within the functions to determine whether to execute TCP or OSI service functionality.

7.1 Association Handling Functions Using ACSE-Services

NOTE: The explanations in this section assume that where there are decision statements in the code, the OSI route is always taken.

7.1.1 DbvAssociateRequest() Function

Identification:

```
int DbvAssociateRequest(      DbvAssocId              *plAssocId,
                              DbvSecs                  lSeconds,
                              DbvTarget               *psTargetIdentification,
                              DbvAssociateState       *peAssociateState,
                              DbvAssociateResult     **ppuAssociateResult,
                              DbvError               **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Establishes a connection to the specified target machine. Attempts to create a valid association between the origin and the specified target using **ISODE** library functions.

Functionality:

Using the information supplied in the `psTargetIdentification` variable, a connection to a target machine is attempted. Depending on the value of `lSeconds`, the function will either wait indefinitely, wait for a specified number of seconds, or not wait at all for the successful completion of the connection.

If the function returns *SI_NOTOK* an error occurred.

If the function was successful `plAssocId` will hold the unique identifier for this association. `peAssociateState` will hold the state of the association.

The `ppuAssociateResult` variable will be updated if `peAssociateState` indicates either *Associate_Accepted* or *Associate_Rejected*.

Steps taken:

If no `DBVAssociateList` structure exists, then this is the first call to `DbvAssociateRequest()`. In this case, create the linked list structure which is used to hold details on all associations.

See if the `psTargetIdentification->eTargetNameOrAddress` specifies *Target_Name*. If it does, convert the name into an address by searching the `target_names.tbl/isoentities` files. If the specified name resolves to an OSI address then the name format is kept and later resolved in the function `DBVOsiConnectToHost()` (Read the explanation of this function Section

If the specified target is to be contaced using ACSE services then the function `DBVOsiConnectToHost()` is called.

If the call to the target was successful or still in progress a new, unique association Id is generated, the association is added to the `DBVAssociateList` linked list (for use internal to the API) and the `peAssociateState` parameter is completed according to the state of the association.

Subordinates:

`DBVGetAddressByName()`

`DBVOsiConnectToHost()`

`DBVfill_error_structure()`

`SL_Create()`

`SL_Locate()`

`SL_Add()`

`DBVFree()`

`DBVAllocCheck()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to an OSI network.

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.1.1

Refer to section 10.2.1 - Function `DBVOsiConnectToHost()` - for details on the completion of an OSI association.

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10: Programmer's Guide: ASEs - Chapter 1.2.1 Association Establishment

Processing:

Pseudo code for generating unique association ids:

```
1. Check if value to be assigned to new association is already used by an
association.
2. If is, then id generator++.
3. Produce error if no IDs are free. Leave function. else Goto 1.
3. Assign generated id to the new association (lAssocId).
4. Increment Id generator.
```

Data:

`DbvAssociateState eAssociateStateLocal` = *Associate_Null* - Holds the value relevant `AssociateState` value. This value is copied to the function parameter `peAssociateState` if the return value is *SI_OK*.

`DBVAssociateDetails *psAssociateDetails` - Used to build and store details about the new association. This is an internal variable. If the associate request is successful, then the structure is placed in a linked list. This list is used internally throughout the API to keep track of currently active associations

`DBVAssociateDetails *psGeneratorAssociateDetails` - Used when a new association id is being created to make sure that an association with the id chosen does not already exist.

`static DbvAssocId IdGenerator = 1` - This variable is used to generate the unique association id. Currently the variable is initialised as 1 and is incremented to create a new id.

`int return_value = SI_OK` - Used throughout the function to record the current return value of the function.

`DbvAssociateResult *puAssociateResult` - De-references the function parameter `**ppuAssociateResult` by one level to increase code readability.

Usage:

```
DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSecs                 lSeconds;
DbvTarget               sTargetId;
DbvAssociateState       eAssociate_State = Associate_Null;
DbvAssociateResult     *puAssociateResult = NULL;
char                    pcAddress[32]="ALIBRARY";
DbvError               *psError;

       /* Obviously read a proper name here into pcAddress */
              :
              :

       sTargetId.eTargetNameOrAddress = Target_Name;
       sTargetId.uNameOrAddress.pcTargetName=
              DbvAlloc(strlen(pcAddress)+1); /* Mustn't forget +1 for '\0' */
       strcpy(sTargetId.uNameOrAddress.pcTargetName, pcAddress);

       if(DbvAssociateRequest(&lAssocId,lSeconds, &sTargetId, &eAssociateState,
              &puAssociateResult, &psError) == SI_OK)
       {
              if(eAssociateState == Associate_Pending)
              {
                     /* Complete the Request */
                            :
              }
              if(eAssociateState == Associate_Accepted)
              {
                     /* Association was accepted */
                            :
              }
              if(eAssociateState == Associate_Rejected)
              {
                     /* Association was rejected, why? */
                     PrintDbvAssociateRejectSource(puAssociateResult->psAssociateReject-
                            >peAssociateRejectSource);
                     PrintDbvAssociateRejectResult(puAssociateResult->psAssociateReject-
                            >peAssociateRejectResult);
                     PrintDbvAssociateRejectReason(puAssociateResult->psAssociateReject-
                            >peAssociateRejectReason);
              }
       }
       else
       {
              PrivateErrorHandling(psError);
                     :
                     :
       }
```

7.1.2 DbvReleaseRequest() Function

Identification:

```
int DbvReleaseRequest(  DbvAssocId                lAssocId,
                        DbvSecs                   lSeconds,
                        DbvReleaseState          *peReleaseState,
                        DbvError                **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Initiates the release of the specified association. Attempts to release the valid association between the origin and the specified target using **ISODE** library functions.

Functionality:

After ensuring that the current association is valid, the function attempts to submit an A-RELEASE.REQUEST service primitive using the **ISODE** library function `AcRelRequest()`. Depending on the service-user supplied value `lSeconds`, the function will either time out or block indefinitely. When the function returns the association will be in one of the following states:

Function returned ***OK***:

- The release request was rejected - `peReleaseState` is updated to *Release_Rejected*.
- The release request was accepted - `peReleaseState` is updated to *Release_Accepted*.

Function returned ***NOTOK***:

- The release request timed out - `peReleaseState` is updated to Release_Pending.
- The release request produced a different error - Fill out `ppsError` structure, return *SI_NOTOK*. `peReleaseState` is not updated.

If the return value of the function is *SI_OK* and `peReleaseState` is equal to *Release_Rejected*, then the association is removed from the internal association list `DBVAssociateList`.

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`AcRelRequest() – ISODE function`

`DBVTrace()`

`SL_RemoveCurrent()`

`DBVLeaveFuncTrace()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to an OSI network.
- The supplied association Id must be valid
- The API state machine should not be violated.

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.1.2

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10: Programmer's Guide: ASEs - Chapter 1.2.2 Association Release

Processing:

None.

Data:

`DbvReleaseState eReleaseStateLocal = Release_Null` - The current release state of the association. The value of this variable is copied into `peReleaseState` at the end of the function only if the return value is *SI_OK*.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`DBVAssociateDetails *psAssociateDetails;` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id. If the release is successful the details are removed from the association linked list `DBVAssociateList`.

`DbvProtocolStack eProtocolStack;` - Used to store the protocol stack used by the association.

`struct AcSAPrelease *psAssocControlRelease=NULL;` – This structure is completed by the **ISODE** function `AcRelRequest()` if the release request is successful. It corresponds to an A-RELEASE.CONFIRMATION event.

`struct AcSAPindication *psAssocControlIndication = NULL;` – This structure is updated with diagnostic information if the call to `AcRelRequest()` fails.

`struct AcSAPabort *psAssocControlAbort = NULL;` – Set to point to a structure within the `psAssocControlIndication` structure.

Usage:

```
DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSecs                 lSeconds;
DbvReleaseState         eReleaseState;
DbvError               *psError=NULL;

        /* You must have a valid association by here */
                        :
                        :
        lSeconds = 2; /* Wait for two seconds */
        if(DbvReleaseRequest(lAssocId, lSeconds, &eReleaseState, &psError)
                == SI_NOTOK)
        {
                /* Call failed */
                PrivateErrorHandling(psError);
                        :
                        :
        }
        else
        {
                if(eReleaseState==Release_Pending)
                {
                        if(DbvConfirmReleaseRequest(lAssocId, lSeconds, &eReleaseState,
                                &psError)==SI_NOTOK)
                                :
                                :
                }
                if(eReleaseState==Release_Accepted)
                {
                                :
                                :
                }
                if(eReleaseState==Release_Rejected)
                {
                        /* Abort the association */
                                :
                                :
                }

        }
```

7.1.3 DbvAbortRequest() Function

Identification

```
int DbvAbortRequest(    DbvAssocId          lAssocId,
                        DbvError          **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Aborts the specified association. Attempts to abort the valid association between the origin and the specified target using **ISODE** library functions.

Functionality:

After ensuring that the current association is valid, the function attempts to submit an A-ABORT.REQUEST service primitive using the **ISODE** library function `AcUAbortRequest()`.

If the function returns OK the association will have been aborted. If the function returned NOTOK, then the error structure will be filled out.

If the association has been aborted then it is removed from the internal association list `DBVAssociateList`.

Note that due to the nature of the ISODE function `AcUAbortRequest()` this function will block until the network peer attempts to listen on the association. This should not usually pose a problem unless the peer application/system has hung.

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`AcUAbortRequest() - ISODE function`

`DBVTrace()`

`DBVACSEOSICheckISODEError()`

`SL_RemoveCurrent()`

`DBVLeaveFuncTrace()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to an OSI network.
- The supplied association Id must be valid

- The API state machine should not be violated

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.1.3

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10: Programmer's Guide: ASEs - Chapter 1.2.3 Association Abort

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id. If the abort is successful the details are removed from the association linked list `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`struct AcSAPindication *psAssocControlIndication = NULL` – This structure is updated with diagnostic information if the call to `AcUAbortRequest()` fails.

`struct AcSAPabort *psAssocControlAbort = NULL` – Set to point to a structure within the `psAssocControlIndication` structure.

Usage:

```
DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvError               *psError=NULL;

        /* You must have a valid association by here */
                        :
        if(DbvAbortRequest(lAssocId, &psError)
                == SI_NOTOK)
        {
                printf("AbortRequest call failed\n");
                PrivateErrorHandling(psError);
                        :
        }
```

```
/* Association has been aborted successfully */
        :
```

7.1.4 DbvConfirmAssociateRequest() Function

Identification:

```
int DbvConfirmAssociateRequest( DbvAssocId              lAssocId,
                                DbvSecs                 lSeconds,
                                DbvAssociateState      *peAssociateState,
                                DbvAssociateResult     **ppuAssociateResult,
                                DbvError               **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

To complete a previously initiated Association Request.

Functionality:

The function first checks whether the association is valid and is in state *Associate_Pending*.

Depending on the value of `lSeconds`, the function will either wait indefinitely, wait for a specified number of seconds, or not wait at all for the successful completion of the connection.

If the function was successful `peAssociateState` will hold the state of the association.

The `ppuAssociateResult` variable will be updated if `peAssociateState` indicates either *Associate_Accepted* or *Associate_Rejected*.

Steps taken:

The completion of the association is achieved by selecting the association until it becomes available for **writing** and performing an `AcAsynAssocRequest()` call. Next, the process of selecting the association until it becomes available for **reading** and performing an `AcAsynAssocRequest()` call is repeated until the call is accepted/rejected, the duration specified in `lSeconds` elapses or an error occurs.

If the association has been completed then `ppuAssociateResult` and `peAssociateState` are completed by the function `DBVOsiCompleteDbvAssociateResult()`.

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`DBVfill_error_structure()`

`PSelectMask()` – ISODE function

`DBVTrace()`

`DBVPOSICheckISODEError()`

`xselect() – ISODE function`

`AcAsynRetryRequest() – ISODE function`

`DBVOsiCompleteDbvAssociateResult()`

`DBVLeaveFuncTrace()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to an OSI network
- The supplied association Id must be valid
- The API state machine should not be violated

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.1.4

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10: Programmer's Guide: ASEs - Chapter 1.2.1 Association Establishment, Chapter 1.4 Select Facility.

Processing:

The processing for completing an asynchronous association is specified below:

WHILE - RETURN_VALUE IS OK, SECONDS HAS NOT TIMED OUT AND ASSOCIATION IS NOT COMPLETED.

IF ASSOCIATION IS IN STATE CONNECTING_1 - SELECT THE ASSOCIATION FOR WRITING.

IF ASSOCIATION IS IN STATE CONNECTING_2 - SELECT THE ASSOCIATION FOR READING.

IF THE ASSOCIATION IS NOW READY FOR WRITING/READING - CALL ISODE FUNCTION AcAsynRetryRequest() - RESULT WILL BE - DONE, CONNECTING_1, CONNECTING_2, NOTOK.

CHECK FOR TIMEOUT.

ENDWHILE

Data:

`DbvAssociateState eAssociateStateLocal = Associate_Null;` – Holds the state of the association internally to the function. The value of `peAssociateState` is updated with the value of this variable if the function will be returning *SI_OK*.

`DBVAssociateDetails *psAssociateDetails` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id. If the abort is successful the details are removed from the association linked list `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`struct PSAPindication *psPresentationIndication = NULL;` – This ISODE structure is updated with diagnostic information if a call to the function `PSelectMask()` fails.

`struct AcSAPconnect *psAssocControlConnect = NULL;` – When the function `AcAsynRetryRequest()` completes the association, this structure is filled with information detailing it's status (e.g. Accept/Reject and negotiated contexts).

`struct AcSAPindication *psAssocControlIndication = NULL` – This ISODE structure is updated with diagnostic information if the call to `AcAsynRetryRequest()` fails.

`struct AcSAPabort *psAssocControlAbort = NULL` – Set to point to a structure within the `psAssocControlIndication` structure.

`fd_set sAssocMask, sSetAssocMask;` - Holds the association in a format compatible with the `xselect()` command. After a call to `xselect()` `sAssocMask` must be reset with the value from `sSetAssocMask`.

`int lNfds;` – Used in `PSelectMask()` to tell the function the number of file descriptors being placed in the mask variable (`sSetAssocMask`).

`long lResult;` – Stores the result of calls to `xselect()` and `PSelectMask()`.

`long lSecondsToWait = 1;` – We have to know how long to continue waiting, because waiting for the completion of an association goes through several stages. Set if `lSeconds` is greater than or equal to one.

`long lTimeout = 1;` the xselect commands will pause for a default one second. Value is set to *OK* (0) if call is *CALL_AYNCHRONOUS*.

Usage:

```
DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSecs                 lSeconds = 1;
DbvAssociateState       eAssociate_State = Associate_Null;
DbvAssociateResult     *puAssociateResult = NULL;
DbvError               *psError=NULL;
int                     return_value=SI_OK;
                :
        /* Attempt an asynchronous AssociateRequest() here */
                :
        while(return_value == SI_OK && aAssociateState == Associate_Pending)
        {
                printf("Association is still pending\n");
```

```
                if((return_value=DbvConfirmAssociateRequest(lAssocId, lSeconds, &eAssociateState,
&puAssociateResult, &psError)) != SI_OK)
                {
                        printf("DbvConfirmAssociateRequest() call failed\n");
                        PrivateErrorHandling(psError);
                                :
                }
        }
        if(return_value == SI_OK && eAssociateState == Associate_Accepted)
        {
                /* Association was accepted */
                        :
        }
        if(return_value == SI_OK && eAssociateState == Associate_Rejected)
        {
                /* Association was rejected, why? */
                PrintDbvAssociateRejectSource(puAssociateResult->psAssociateReject-
                        >peAssociateRejectSource);
                PrintDbvAssociateRejectResult(puAssociateResult->psAssociateReject-
                        >peAssociateRejectResult);
                PrintDbvAssociateRejectReason(puAssociateResult->psAssociateReject-
                        >peAssociateRejectReason);
        }
```

7.1.5 DbvConfirmReleaseRequest() Function

Identification:

```
int DbvConfirmReleaseRequest( DbvAssocId          lAssocId,
                              DbvSecs             lSeconds,
                              DbvReleaseState    *peReleaseState,
                              DbvError          **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Completes a previously initiated Release Request.

Functionality:

First the supplied association is checked to ensure that it is in state *Release_Pending*. The ISODE function `AcRelRetryRequest()` is called for the association specified. The value of `lSeconds` is converted to the ISODE equivalent and also supplied to the function. If the `AcRelRetryRequest()` function returns *NOTOK* and the reason is *ACS_TIMER*, then the state of the association is still *Release_Pending*. The function returning *OK* indicates that the release was accepted or rejected. If the return value is *SI_OK*, the `peReleaseState` is updated with the new state. If the association is now released, the association is removed from the internal association list `DBVAssociateList`.

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`SL_Current()`

`AcRelRetryRequest() - ISODE function`

`DBVfill_error_structure()`

`SL_RemoveCurrent()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to an OSI network
- The supplied association Id must be valid
- The API state machine should not be violated

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.1.5

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10: Programmer's Guide: ASEs - Chapter 1.2.2 Association Release.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id. If the abort is successful the details are removed from the association linked list `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`DbvReleaseState eReleaseStateLocal; = Holds the release state of the association internally. Only if the return value of the function is SI_OK, the value of this variable copied into eReleaseState.`

`struct AcSAPrelease *psAssocControlRelease=NULL;`

`struct AcSAPindication *psAssocControlIndication = NULL` – This ISODE structure is updated with diagnostic information if the call to `AcAsynRetryRequest()` fails.

`struct AcSAPabort *psAssocControlAbort = NULL` – Set to point to a structure within the `psAssocControlIndication` structure.

Usage:

`psAssocControlIndication` structure.

Usage:

```
DbvAssocId             lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSecs                lSeconds;
DbvReleaseState        eReleaseState;
DbvError              *psError=NULL;
```

```
/* You must have done a DbvReleaseRequest by here */
                :
                :
lSeconds = 2; /* Wait for two seconds */
while(return_value == SI_OK && eReleaseState == Release_Pending)
{
        if((return_value = DbvConfirmReleaseRequest(lAssocId, lSeconds, &eReleaseState,
                &psError)) == SI_NOTOK)
        {
                /* Call failed */
                PrivateErrorHandling(psError);
                        :
                        :
        }
}
if(eReleaseState==Release_Accepted)
{
                :
                :
}
if(eReleaseState==Release_Rejected)
{
        /* Abort the association */
                :
                :
}
```

7.2 Association Handling Functions Using TCP-Services

NOTE: The explanations in this section assume that where there are decision statements in the code, the TCP route is always taken.

7.2.1 DbvAssociateRequest() Function

Identification:

The identification of the function is:

```
int DbvAssociateRequest(DbvAssocId              *plAssocId,
                        DbvSecs                  lSeconds,
                        DbvTarget               *psTargetIdentification,
                        DbvAssociateState       *peAssociateState,
                        DbvAssociateResult     **ppuAssociateResult,
                        DbvError               **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to establish a connection to the specified target machine.

Functionality:

Using the information supplied in the `psTargetIdentification` variable, a connection to a target machine is attempted.

Depending on the value of `lSeconds`, the function will either wait indefinitely, wait for a specified number of seconds, or not wait at all for the successful completion of the connection.

If the function returns *`SI_NOTOK`* an error occurred and the DbvError structure will contain relevant diagnostic information.

If the request was successful `plAssocId` will hold the unique identifier for this association. `peAssociateState` will hold *`Associate_Accepted`* . If the request has not yet been completed, `peAssociateState` will hold *`Associate_Pending`*. When using TCP services, it is not possible for *`Associate_Reject`* to occur.

The `ppuAssociateResult` variable will be updated if the call was successful, although the parameters `peApplicationProtocol` and `ppeAbstractSyntax` will always be set to *`Unknown_Protocol`* and *`No_Syntaxes_Provided`* respectively.

This function will add a new `DBVAssociateDetails` structure to the global linked list `DBVAssociateList`.

Subordinates:

The subordinates to this function are:

`SL_Locate()`

```
SL_Add()
DBVAllocCheck()
DBVAlloc()
DBVGetAddressByName()
DBVTcpConnectToHost()
```

Dependencies:

The dependencies of this function are:

- The machine must be connected to a TCP/IP network.

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- TCP/IP Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.2.1

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

Pseudo code for generating unique association ids:

```
1. Check if value to be assigned to new association is already used by an
association.
2. If is, then id generator++.
3. Produce error if no IDs are free. Leave function. else Goto 1.
3. Assign generated id to the new association (lAssocId).
4. Increment Id generator.
```

Data:

`DbvAssociateState eAssociateStateLocal` = *Associate_Null* - Holds the relevant AssociateState value. This value is copied to the function parameter `peAssociateState` if the return value is *SI_OK*.

`DBVAssociateDetails *psAssociateDetails` - Used to build and store details about the new association. This is an internal variable. If the associate request is successful, then the structure is placed in a linked list. This list is used internally throughout the API to keep track of currently active associations

`DBVAssociateDetails *psGeneratorAssociateDetails` - Used when a new association Id is being created to make sure that an association with the Id chosen does not already exist.

`static DbvAssocId IdGenerator = 1` - This variable is used to generate the unique association Id. Currently the variable is initialised as 1 and is incremented to create a new Id.

`int return_value = SI_OK` - Used throughout the function to record the current return value of the function.

`DbvAssociateResult *puAssociateResult` - De-references the function parameter `**ppuAssociateResult` by one level to increase code readability.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget              *psTargetId = (DbvTarget*)NULL;
DbvAssociateState       eAssocState = Associate_Null;
DbvAssociateResult     *puAssociateResult = NULL;
DbvError               *psError = NULL;
       .
       .
       .
       psTargetId.eTargetNameOrAddress = Target_Address;
       psTargetId.uNameOrAddress.psTargetInformation = (struct DbvTargetInformation *)
              DbvAlloc(sizeof(struct DbvTargetInformation));
       psTargetId.uNameOrAddress.psTargetInformation->eOsiOrInternetAddress = Internet_Address;
       psTargetId.uNameOrAddress.psTargetInformation->uOsiOrInternetAddress.psInternetAddress =
              (struct DbvInternetAddress *) DbvAlloc(sizeof(struct DbvInternetAddress));
       psTargetId.uNameOrAddress.psTargetInformation->uOsiOrInternetAddress.psInternetAddress
              ->ulPortNumber = port;
       psTargetId.uNameOrAddress.psTargetInformation->uOsiOrInternetAddress.psInternetAddress
              ->pcHostAddress = host_address;

if (DbvAssociateRequest(&lAssocId, CALL_BLOCKING, psTargetId, &eAssocState,
    &puAAssociateResult, &psError) == SI_OK)
{
   if (eAssocState == Associate_Accepted)
   {
      /*
          Inspect puAssociateResult->psProtocolAndSyntaxes
      */
          FreeAssociateResult(Associate_Accepted, &puAssociateResult);
   }
   else if(eAssocState == Associate_Pending)
   {
       /* Complete the association */
   }
   else
   {
       /* bug in function DbvAssociateRequest() (for TCP) */
   }
}
else
{
   PrivateErrorHandling(psError);
                :
                :
}

FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
```

7.2.2 DbvReleaseRequest() Function

Identification:

The identification of the function is:

```
int DbvReleaseRequest(  DbvAssocId          lAssocId,
                        DbvSecs             lSeconds,
                        DbvReleaseState    *peAReleaseState,
                        DbvError          **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this utilitiy is to release the specified association (uses the TCP service `close()`).

Functionality:

This function extracts the socket file descriptor from the `DBVAssociateDetails` structure corresponding to the given association Id. This file descriptor is then closed. This means that in the context of a TCP association there is no release negotiation with the target. In the context of TCP services, the value of *lSeconds* is ignored. If the `close()` is successful the association id is removed from the list and the function returns *SI_OK*. The function parameter `peReleaseState` will always return *Release_Accepted* for TCP services.

If the `close()` is not successful the `DbvError` structure will be completed. The contents of the structure will indicate whether the association still exists.

This function will produce an error if the internal state machine is violated (see Section 5).

If the return value is *`SI_OK`* the function will remove and free the `psAssociateDetails` structure corresponding to the association Id specified, from the global linked list `DBVAssociateList`.

Subordinates:

The subordinates to this function are:

```
int SIWhichStack()
SL_Current()
DBVfill_error_structure()
SL_RemoveCurrent()
DBVEnterFuncTrace()
DBVLeaveFuncTrace()
```

Dependencies:

The dependencies of this function are:

- There must be no outstanding responses to sent requests (which would violate the state machine - See section 5).
- The association should exist.

Interfaces:

This function effects the following global variables:

- The function uses the `DBVAssociateList` global linked list variable.

Resources:

This function will use the following resources:

- TCP/IP Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.2.2

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

The function will query the association's state machine to see if a release is valid.

Data:

`DbvReleaseState eAReleaseStateLocal` = *Release_Null* - Used to store the value throught the life of the function. The output parameter `*peAReleaseState` is updated only if the function is going to return *SI_OK*.

`DBVAssociateDetails *psAssociateDetails` - Used to hold the associate details for the association specified in `lAssocId`.

`DbvProtocolStack eProtocolStack` - Used to hold the protocol used by the association (*TCP_Stack* or *OSI_Stack*).

`int return_value = SI_OK` - Is used throughout the function to record the current retutrn value of the function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
DbvAssocId                      lAssocId;
DbvReleaseState                 eReleaseState = Release_Null;
DbvError                       *psError = NULL;

if (DbvReleaseRequest(lAssocId, CALL_BLOCKING, &eReleaseState, &psError) == SI_OK)
{
   if (eReleaseState == Release_Accepted)
   {
      /* release buffer, terminate process */
   }
   else
   {
       /* bug in function DbvReleaseRequest() */
   }
}
else
{
```

```
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
}
```

7.2.3 DbvAbortRequest() Function

Identification:

The identification of the function is:

```
int DbvAbortRequest(    DbvAssocId          lAssocId,
                        DbvError          **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

The purpose of this function is to abort the the association using the TCP service `close()`.

Functionality:

This function extracts the socket file descriptor from the `DBVAssociateDetails` structure corresponding to the given association Id. This file descriptor is then closed. If the `close()` is successful the association id is removed from the list and the function returns *`SI_OK`*.

If an error occurs while closing the socket the `DbvError` structure will be completed with relevant diagnostic information.

If the return value is *`SI_OK`*, the function will remove and free the `psAssociateDetails` structure corresponding to the association id specified, from the global linked list `DBVAssociateList`.

Subordinates:

The subordinates to this function are:

```
int SIWhichStack()
SL_Current()
DBVfill_error_structure()
SL_RemoveCurrent()
DBVEnterFuncTrace()
DBVLeaveFuncTrace()
```

Dependencies:

The dependencies of this function are:

- The association should exist

Interfaces:

None

Resources:

This function will use the following resources:

- TCP/IP Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.2.3

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

N/A

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the associate details for the association specified in `lAssocId`.

`ProtocolStack eProtocolStack` - Used to hold the protocol used by the association (*TCP_Stack* or *OSI_Stack*).

`int return_value = SI_OK` - Is used throughout the function to record the current retutrn status of the function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
DbvAssocId      lAssocId;
DbvError       *psError = NULL;

if (DbvAbortRequest(lAssocId,&psError) == SI_OK)
{
   /* association aborted */

                     :
                     :
}
else
{
   /* if psError->eAssociationState == Association_Open
      the association still exists
   */

   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
```

7.2.4 DbvConfirmAssociateRequest() Function

Identification:

The identification of the function is:

```
int DbvConfirmAssociateRequest(     DbvAssocId               lAssocId,
                                    DbvSecs                  lSeconds,
                                    DbvAssociateState       *peAssociateState,
                                    DbvAssociateResult     **ppuAssociateResult,
                                    DbvError               **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Ask for the completion of a previously initialied connection establishment.

Functionality:

This function first checks to see whether the state of the association is *Associate_Pending.* If so the function tries to complete the connection for the duration specified in `lSeconds`. This is done using the Unix TCP functions `select()` and `connect()`.

If the request was successful `peAssociateState` will hold *Associate_Accepted* . If the request has not yet been completed, `peAssociateState` will hold *Associate_Pending*. When using TCP services, it is not possible for *Associate_Reject* to occur.

The `ppuAssociateResult` variable will be updated if the call was successful, although the parameters `peApplicationProtocol` and `ppeAbstractSyntax` will always be set to *Unknown_Protocol* and *No_Syntaxes_Provided* respectively.

If this function is called after a successful `AssociateRequest()` call the `DbvError` structure will be completed with a diagnostic message stating that the association is already established.

Subordinates:

`DBVCompleteTCPAssociation()`

`int SIWhichStack()`

`SL_Current()`

`DBVfill_error_structure()`

`DBVEnterFuncTrace()`

`DBVLeaveFuncTrace()`

Dependencies:

The association must still be pending.

Interfaces:

N/A

Resources:

This function will use the following resources:

- TCP/IP Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.1.2.4

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Processing:

N/A

Data:

`DbvAssociateState eAssociateStateLocal` = *Associate_Null* - Holds the relevant AssociateState value. This value is copied to the function parameter peAssociateState if the return value is *SI_OK*.

`DBVAssociateDetails *psAssociateDetails` - Used to hold the associate details for the association specified in `lAssocId`.

`ProtocolStack eProtocolStack` - Used to hold the protocol used by the association (*TCP_Stack* or *OSI_Stack*).

`int return_value` = *SI_OK* - Is used throughout the function to record the current retutrn status of the function.

`DbvAssociateResult *puAssociateResult` - De-references the function parameter `**ppuAssociateResult` by one level to increase code readability.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
DbvAssocId              lAssocId;
DbvError               *psError = NULL;
DbvAssociateState       eAssociateState;
DbvAssociateResult     *puAssociateResult=NULL;

        :
/* Assume a call to DbvAssociateRequest() has returned Associate_Pending */
        :
if(DbvConfirmAssociateRequest(lAssocId, lSeconds, &eAssociateState, &puAssociateResult,
        &psError) != SI_OK)
{
        printf("An error ocurred in ConfirmAssociateRequest\n");
}
if (eAssociateState == Associate_Accepted)
{
        /*
        Inspect puAssociateResult->psProtocolAndSyntaxes
        */
}
```

```
if(eAssocState == Associate_Pending)
{
        /* Complete the association */
}
        :

        :
```

7.2.5 DbvConfirmReleaseRequest() Function

Identification:

```
int DbvConfirmReleaseRequest( DbvAssocId          lAssocId,
                              DbvSecs             lSeconds,
                              DbvReleaseState    *peReleaseState,
                              DbvError          **ppsError );
```

Type:

This function is in source code file **originassoc.c** which itself is contained in the library **libdbvorigin**.

Purpose:

Ask for the completion of a previously initiated connection release.

Function:

For a TCP connection this function will never need to be called. This is because the `DbvReleaseRequest()` function will never return *`Release_Pending`*.

When the function is entered the association's `psAssociateDetails` structure is queried for it's current state. If it is not *`Release_Pending`* (which it never will be for a TCP association!) the `ppsError` structure is completed with the appropriate error and *`SI_NOTOK`* is returned.

Subordinates:

N/A

Dependencies:

N/A

Interfaces:

N/A

Resources:

N/A

References:

Additional information on this function can be found in:

Section 7.1.5

D-008 API-Definition, 3.1.2.5

Processing:

N/A

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the associate details for the association specified in `lAssocId`.

`ProtocolStack eProtocolStack` - Used to hold the protocol used by the association (*TCP_Stack* or *OSI_Stack*).

Usage:

N/A

7.4 SR-/Z39.50 Functions

7.4.1 DbvInitializeRequest() Function

Identification:

```
int DbvInitializeRequest(     DbvAssocId          lAssocId,
                              DbvInitializeReq   *psInitializeReq,
                              DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_init.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 InitializeRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psInitializeReq structure are checked by calling subordinate function DBVCheckDbvInitReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows an InitializeRequest PDU. If sending an InitializeRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psInitializeReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_initrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.1

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvInitializeRequest(DbvAssocId lAssocId, DbvInitializeReq
*psInitializeReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;
```

```
      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVInitializeRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psInitializeReq are valid */
      IF (DBV_CHECK_DBV_INITREQ_FIELDS(lAssocId, psInitializeReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Init Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_initrequest_structure;

            /* Map fields in psInitializeReq structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_INITREQUEST(PDU_ptr, psInitializeReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function trracing */
      DBV_LEAVE_FUNC_TRACE("DbvInitializeRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc InitializeRequest PDU structure fields (DBVsetup_snacc_intrequest()) and Snacc InitializeRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psInitializeReq fields, the association and the state machine state are all valid for an InitializeRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psInitializeReq fields onto Snacc InitializeRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc InitializeRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

In ANSI C, the setjmp() and longjmp() macros have the following syntax:

```
#include<setjmp.h>

int setjmp(jump_buf env);

void longjmp(jmp_buf env, int val);
```

The setjmp() macro saves program state information in env for use by a future longjmp(). The return is zero from a direct call of setjmp() and non-zero from a subsequent call of longjmp().

Any longjmp() restores the program state saved by the most recent call to setjmp() using information saved in the env parameter. Execution resumes as if the setjmp() function has just executed and returned the non-zero value val. `jmp_buf env` - Variable used to hold the environment of the program at the moment of the setjmp() call. If a longjmp() call is made from a subordinate function

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvInitializeReq       *psInitializeReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psInitializeReq and fill in structure.
*/

if (DbvInitializeRequest(lAssocId, psInitializeReq, &psError) == SI_OK)
{
```

```
   FreeInitializeReq(&psInitializeReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeInitializeReq(&psInitializeReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.2 DbvSearchRequest() Function

Identification:

```
int DbvSearchRequest(   DbvAssocId          lAssocId,
                        DbvSearchReq       *psSearchReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_search.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 SearchRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psSearchReq structure are checked by calling subordinate function DBVCheckDbvSearchReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a SearchRequest PDU. If sending a SearchRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psSearchReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_srchrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.2

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvSearchRequest(DbvAssocId lAssocId, DbvSearchReq *psSearchReq,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVSearchRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psSearchReq are valid */
```

```
      IF (DBV_CHECK_DBV_SRCHREQ_FIELDS(lAssocId, psSearchReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Srch Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_srchrequest_structure;

            /* Map fields in psSearchReq structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_SRCHREQUEST(PDU_ptr, psSearchReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function trracing */
      DBV_LEAVE_FUNC_TRACE("DbvSearchRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc SearchRequest PDU structure fields (DBVsetup_snacc_srchrequest()) and Snacc SearchRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psSearchReq fields, the association and the state machine state are all valid for a SearchRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psSearchReq fields onto Snacc SearchRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc SearchRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSearchReq           *psSearchReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psSearchReq and fill in structure.
*/

if (DbvSearchRequest(lAssocId, psSearchReq, &psError) == SI_OK)
{
   FreeSearchReq(&psSearchReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeSearchReq(&psSearchReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.3 DbvPresentRequest() Function

Identification:

```
int DbvPresentRequest(  DbvAssocId          lAssocId,
                        DbvPresentReq      *psPresentReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_present.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 PresentRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psPresentReq structure are checked by calling subordinate function DBVCheckDbvPresReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a PresentRequest PDU. If sending a PresentRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psPresentReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_presrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.3

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvPresentRequest(DbvAssocId lAssocId, DbvPresentReq
*psPresentReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVPresentRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psPresentReq are valid */
```

```
      IF (DBV_CHECK_DBV_PRESREQ_FIELDS(lAssocId, psPresentReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Pres Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_presrequest_structure;

            /* Map fields in psPresentReq structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_PRESREQUEST(PDU_ptr, psPresentReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvPresentRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc PresentRequest PDU structure fields (DBVsetup_snacc_presrequest()) and Snacc PresentRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psPresentReq fields, the association and the state machine state are all valid for a PresentRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psPresentReq fields onto Snacc PresentRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc PresentRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvPresentReq          *psPresentReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psPresentReq and fill in structure.
*/

if (DbvPresentRequest(lAssocId, psPresentReq, &psError) == SI_OK)
{
   FreePresentReq(&psPresentReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreePresentReq(&psPresentReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.4 DbvDeleteResultSetRequest() Function

Identification:

```
int DbvDeleteResultSetRequest(      DbvAssocId          lAssocId,
                                    DbvDeleteResultSetReq
*psDeleteResultSetReq,
                                    DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_resrep.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 DeleteResultSetRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psDeleteResultSetReq structure are checked by calling subordinate function DBVCheckDbvDelReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a DeleteResultSetRequest PDU. If sending a DeleteResultSetRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psDeleteResultSetReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_delrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.4

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvDeleteResultSetRequest(DbvAssocId lAssocId,
DbvDeleteResultSetReq *psDeleteResultSetReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVDeleteResultSetRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psDeleteResultSetReq are valid */
```

```
      IF (DBV_CHECK_DBV_DELREQ_FIELDS(lAssocId, psDeleteResultSetReq, ppsError)
THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the DeleteResultSet Request is valid and can be sent to a valid
associated target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_delrequest_structure;

            /* Map fields in psDeleteResultSetReq structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_DELREQUEST(PDU_ptr, psDeleteResultSetReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvDeleteResultSetRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc DeleteResultSetRequest PDU structure fields (DBVsetup_snacc_delrequest()) and Snacc DeleteResultSetRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psDeleteResultSetReq fields, the association and the state machine state are all valid for a DeleteResultSetRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psDeleteResultSetReq fields onto Snacc DeleteResultSetRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc DeleteResultSetRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvDeleteResultSetReq         *psDeleteResultSetReq = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psDeleteResultSetReq and fill in structure.
*/

if (DbvDeleteResultSetRequest(lAssocId, psDeleteResultSetReq, &psError) == SI_OK)
{
   FreeDeleteResultSetReq(&psDeleteResultSetReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeDeleteResultSetReq(&psDeleteResultSetReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.5 DbvScanRequest() Function

Identification:

```
int DbvScanRequest(     DbvAssocId          lAssocId,
                        DbvScanReq         *psScanReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_scan.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 ScanRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psScanReq structure are checked by calling subordinate function DBVCheckDbvScanReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a ScanRequest PDU. If sending a ScanRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psScanReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_scanrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.5

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvScanRequest(DbvAssocId lAssocId, DbvScanReq *psScanReq,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVScanRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psScanReq are valid */
```

```
      IF (DBV_CHECK_DBV_SCANREQ_FIELDS(lAssocId, psScanReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Scan Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_scanrequest_structure;

            /* Map fields in psScanReq structure onto fields in SNACC structure
*/
            DBV_SETUP_SNACC_DELREQUEST(PDU_ptr, psScanReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvScanRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ScanRequest PDU structure fields (DBVsetup_snacc_scanrequest()) and Snacc ScanRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psScanReq fields, the association and the state machine state are all valid for a ScanRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psScanReq fields onto Snacc ScanRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ScanRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvScanReq                    *psScanReq = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psScanReq and fill in structure.
*/

if (DbvScanRequest(lAssocId, psScanReq, &psError) == SI_OK)
{
   FreeScanReq(&psScanReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeScanReq(&psScanReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.6 DbvResourceReportRequest() Function

Identification:

```
int DbvResourceReportRequest( DbvAssocId          lAssocId,
                              DbvResourceReportReq
*psResourceReportReq,
                              DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_reprep.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 ResourceReportRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psResourceReportReq structure are checked by calling subordinate function DBVCheckDbvResRepReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a ResourceReportRequest PDU. If sending a ResourceReportRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psResourceReportReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_resreprequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.6

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvResourceReportRequest(DbvAssocId lAssocId,
DbvResourceReportReq *psResourceReportReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVResourceReportRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psResourceReportReq are valid */
```

```
      IF (DBV_CHECK_DBV_RESREPREQ_FIELDS(lAssocId, psResourceReportReq,
ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the ResourceReport Request is valid and can be sent to a valid
associated target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_resreprequest_structure;

            /* Map fields in psResourceReportReq structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_RESREPREQUEST(PDU_ptr, psResourceReportReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvResourceReportRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ResourceReportRequest PDU structure fields (DBVsetup_snacc_resreprequest()) and Snacc ResourceReportRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psResourceReportReq fields, the association and the state machine state are all valid for a ResourceReportRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psResourceReportReq fields onto Snacc ResourceReportRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ResourceReportRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvResourceReportReq          *psResourceReportReq = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psResourceReportReq and fill in structure.
*/

if (DbvResourceReportRequest(lAssocId, psResourceReportReq, &psError) == SI_OK)
{
   FreeResourceReportReq(&psResourceReportReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeResourceReportReq(&psResourceReportReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.7 DbvSortRequest() Function

Identification:

```
int DbvSortRequest(     DbvAssocId          lAssocId,
                        DbvSortReq         *psSortReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_sort.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 SortRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psSortReq structure are checked by calling subordinate function DBVCheckDbvSortReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a SortRequest PDU. If sending a SortRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psSortReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_sortrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.7

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvSortRequest(DbvAssocId lAssocId, DbvSortReq *psSortReq,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVSortRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psSortReq are valid */
```

```
      IF (DBV_CHECK_DBV_SORTREQ_FIELDS(lAssocId, psSortReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Sort Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_sortrequest_structure;

            /* Map fields in psSortReq structure onto fields in SNACC structure
*/
            DBV_SETUP_SNACC_SORTREQUEST(PDU_ptr, psSortReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvSortRequest", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc SortRequest PDU structure fields (DBVsetup_snacc_sortrequest()) and Snacc SortRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psSortReq fields, the association and the state machine state are all valid for a SortRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psSortReq fields onto Snacc SortRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc SortRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSortReq             *psSortReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psSortReq and fill in structure.
*/

if (DbvSortRequest(lAssocId, psSortReq, &psError) == SI_OK)
{
   FreeSortReq(&psSortReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeSortReq(&psSortReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.8 DbvExtendedServicesRequest() Function

Identification:

```
int DbvExtendedServicesRequest(     DbvAssocId          lAssocId,
                                    DbvExtendedServicesReq
*psExtendedServicesReq,
                                    DbvError          **ppsError);
```

Type:

This function is in source code file **origin_z3950_extend.c** which itself is contained in the library **libdbvorigin**

Purpose:

The purpose of this function is to send a Z39.50 ExtendedServicesRequest PDU to the target of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psExtendedServicesReq structure are checked by calling subordinate function DBVCheckDbvExtendReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a ExtendedServicesRequest PDU. If sending a ExtendedServicesRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psExtendedServicesReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_extendrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.8

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvExtendedServicesRequest(DbvAssocId lAssocId,
DbvExtendedServicesReq *psExtendedServicesReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVExtendedServicesRequest", lAssocId);

      return_value =  FALSE;

      /* If fields in psExtendedServicesReq are valid */
```

```
      IF (DBV_CHECK_DBV_EXTENDREQ_FIELDS(lAssocId, psExtendedServicesReq,
ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the ExtendedServices Request is valid and can be sent to a valid
associated target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_extendrequest_structure;

            /* Map fields in psExtendedServicesReq structure onto fields in
SNACC structure */
            DBV_SETUP_SNACC_EXTENDREQUEST(PDU_ptr, psExtendedServicesReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvExtendedServicesRequest", lAssocId,
return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ExtendedServicesRequest PDU structure fields (DBVsetup_snacc_extendrequest()) and Snacc ExtendedServicesRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psExtendedServicesReq fields, the association and the state machine state are all valid for an ExtendedServicesRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psExtendedServicesReq fields onto Snacc ExtendedServicesRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ExtendedServicesRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvExtendedServicesReq        *psExtendedServicesReq = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psExtendedServicesReq and fill in structure.
*/

if (DbvExtendedServicesRequest(lAssocId, psExtendedServicesReq, &psError) == SI_OK)
{
   FreeExtendedServicesReq(&psExtendedServicesReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeExtendedServicesReq(&psExtendedServicesReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.9 DbvCloseRequest() Function

Identification:

```
int DbvCloseRequest(    DbvAssocId          lAssocId,
                        DbvCloseReq        *psCloseReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **shared_z3950_close.c** which itself is contained in the library **libdbv**

Purpose:

The purpose of this function is to send via a subordinate function a Z39.50 Close PDU to the process at the other end of the communication virtual circuit in the context of a 'CloseRequest'.

Function:

a) Function tracing is enabled.

b) Function DBVClose() is called. In Z39.50 there is only a Close PDU, there is no CloseRequest/CloseResponse pair. Subordinate DBVClose(), which sets up and sends a Close PDU to the other end of the communication virtual circuit is called by both **DbvCloseRequest()** and **DbvCloseResponse()** .

c) Function tracing is disabled.

Subordinates:

DBVClose(). Please see the DBVClose() description below.

Dependencies:

Please see the DBVClose() description below.

Interfaces:

Please see the DBVClose() description below.

Resources:

Please see the DBVClose() description below.

References:

Please see the DBVClose() description below.

Processing:

Please see the DBVClose() description below.

Data:

int return_value - Integer variable assigned the value returned by subordinate function DBVClose() which indicates if a Close PDU has been successfully encoded and sent to the other end of the communication virtual circuit. **return_value** is passed to the trace disabling call at the end of the function.

Please see the DBVClose() description below.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvCloseReq            *psCloseReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psCloseReq and fill in structure.
*/

if (DbvCloseRequest(lAssocId, psCloseReq, &psError) == SI_OK)
{
   FreeCloseReq(&psCloseReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeCloseReq(&psCloseReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.9.1 DBVClose() Function - A Subordinate Of Functions DbvCloseRequest() And DbvCloseResponse()

Identification:

```
int DBVClose(     DbvAssocId          lAssocId,
                  DbvCloseReq        *psCloseReq,
                  DbvError          **ppsError);
```

Type:

This function is in source code file **shared_z3950_close.c** which itself is contained in the library **libdbv**

Purpose:

The purpose of this function is to send a Z39.50 CloseRequest PDU to the process at the other end of the communication virtual circuit.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psCloseReq structure are checked by calling subordinate function DBVCheckDbvCloseReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a CloseRequest PDU. If sending a CloseRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psCloseReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_closerequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value of variable return_variable is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the target must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvClose(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError
**ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      return_value =  FALSE;

      /* If fields in psCloseReq are valid */
      IF (DBV_CHECK_DBV_CLOSEREQ_FIELDS(lAssocId, psCloseReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN
```

```
                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Close Request is valid and can be sent to a valid associated
target */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_closerequest_structure;

            /* Map fields in psCloseReq structure onto fields in SNACC structure
*/
            DBV_SETUP_SNACC_CLOSEREQUEST(PDU_ptr, psCloseReq);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc CloseRequest PDU structure fields (DBVsetup_snacc_closerequest()) and Snacc CloseRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psCloseReq fields, the association and the state machine state are all valid for a CloseRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psCloseReq fields onto Snacc CloseRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc CloseRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

`jmp_buf env` - Program state variable used by the setjmp() macro call and any subsequent longjmp() macro calls.

7.4.10 DbvCloseResponse() Function

Identification:

```
int DbvCloseResponse(   DbvAssocId          lAssocId,
                        DbvCloseReq        *psCloseRsp,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **shared_z3950_close.c** which itself is contained in the library **libdbv**

Purpose:

The purpose of this function is to send via a subordinate function a Z39.50 Close PDU to the process at the other end of the communication virtual circuit in the context of a 'CloseResponse'.

Function:

a) Function tracing is enabled.

b) Function DBVClose() is called. In Z39.50 there is only a Close PDU, there is no CloseRequest/CloseResponse pair. Subordinate DBVClose(), which sets up and sends a Close PDU to the other end of the communication virtual circuit is called by both **DbvCloseRequest()** and **DbvCloseResponse()** .

c) Function tracing is disabled.

Subordinates:

DBVClose(). Please see the DBVClose() description above.

Dependencies:

Please see the DBVClose() description above.

Interfaces:

Please see the DBVClose() description above.

Resources:

Please see the DBVClose() description above.

References:

Please see the DBVClose() description above.

Processing:

Please see the DBVClose() description above.

Data:

`int return_value` - Integer variable assigned the value returned by subordinate function DBVClose() which indicates if a Close PDU has been successfully encoded and sent to the other end of the communication virtual circuit. **`return_value`** is passed to the trace disabling call at the end of the function.

Please see the DBVClose() description below.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvCloseRsp            *psCloseRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psCloseRsp and fill in structure.
*/

if (DbvCloseResponse(lAssocId, psCloseRsp, &psError) == SI_OK)
{
   FreeCloseRsp(&psCloseRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeCloseRsp(&psCloseRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

7.4.11 ReceiveDataOrigin() Function

Identification:

```
int ReceiveDataOrigin(  DbvAssocId          lAssocId,
                        DbvSecs             lSeconds,
                        DbvOriginData     **ppsOriginData,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **origin_Z3950_main.c** and is contained in library **libdbvorigin**.

Purpose:

Waits for a complete PDU of Z39.50 data to be sent by the target, decodes the data then sets ppsOriginData appropriately to contain the PDU structure format decoded from the data.

Functionality:

a) Function tracing is invoked by calling DBVEnterFuncTrace().

b) The association is checked to ensure that is it valid by calling SIWhichStack(). If the association is valid, variable `return_value` is set to *SI_OK*, if it is invalid `return_value` is set to *SI_NOTOK*.

c) If the assocition is valid (`return_value` set to *SI_OK*) then a complete PDU of data is received from the target and converted into a Snacc PDU structure by calling either

c1) DBVreceive_data_over_socket() if TCP/IP communication is being used by the current association

c2) DBVreveive_data_over_OSI() if OSI communication is being used by the current association

If 'unknown' Z39.50 PDU data is received (that of a PDU not defined in the sitypes.h file), the received BER data is not decoded but entered into a special Snacc PDU called UNKNOWN_PDU as a char* and length pair. The UNKNOWN_PDU is unique to DBV-OSI II, it was defined in ASN.1 and created by SIL sprecifically for this project.

If an error occurs during data reception then the receive function being used returns with *SI_NOTOK*.

d) If data for a complete Snacc known or unknown PDU has been received by DBVreceive_data_over_socket()/DBVreceive_data_over_OSI, function DBVconvert_PDU_to_origin_data() is called to map the Snacc PDU data onto structures in the ppsOriginData structure (see **Section 10.10** - PDU Checking And Mapping Subordinate Functions). Any field allocing errors in DBVconvert_PDU_to_origin_data() or any of its subordinates are caught and trapped by a setjmp(), longjmp() mechanism and result in the error structure passed as a parameter to the function being completed with diagnostic information. An UNKNOWN_PDU is mapped onto the psInvalidAPDU structure. Variable `return_value` is set to *SI_OK* if all mapping succeeds or *SI_NOTOK* if mapping fails. An

e) Function tracing is turned off.

Subordinates:

The subordinates to this function are:

`DBVEnterFuncTrace()`

`DBVLeaveFuncTrace()`

`SL_Current()`

`SIWhichStack()`

`DBVreceive_data_over_socket()`

`DBVreceive_data_over_osi()`

`DBVconvert_pdu_to_origin_data()`

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association should be valid.
- The Z39.50 state machine must be satisfied.

Interfaces:

This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, Section 3.1.3.11

Global variable information - Section 11 of this document.

Internal function information - Section 10 of this document.

Z39.50 internal state machine - Section 5 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

```
FUNCTION DbvReceiveDataOrigin( DbvAssocId lAssocId, DbvSecs lSeconds,
DbvOriginData **ppsOriginData, DbvError **ppsError):
```

```
      PDU PDU_to_decode;
      PDU *PDU_ptr;
      int return_value = SI_OK;
      DbvProtocolStack eProtocolStack;

      PDU_ptr = &PDU_to_decode;

      DBV_ENTER_FUNC_TRACE("ReceiveDataOrigin",lAssocId);
      IF  (SIWhichStack(lAssocId,  &eProtocolStack,  ppsError)  ==  SI_OK)  THEN

            eProtocolVersion = STACK_OF_CURRENT_ASSOCIATION();
            IF (eProtocolVersion == TCP) THEN
                        return_value                                          =
DBV_RECEIVE_DATA_OVER_TCP_AND_DECODE_INTO_SNACCPDU(lAssocId, PDU_ptr);
            ELSE
                  return_value                                                =
DBV_RECEIVE_DATA_OVER_OSI_AND_DECODE_INTO_SNACCPDU(lAssocId, PDU_ptr);
            ENDIF
      ENDIF

      IF (return_value == SI_OK) THEN
            return_value  =  DBV_CONVERT_SNACCPDU_TO_ORIGIN_DATA(*ppsOriginData,
PDU_ptr);
      ENDIF

      DBV_LEAVE_FUNC_TRACE("ReceiveDataOrigin",lAssocId, return_value);
      return return_value;
END FUNCTION.
```

Data:

`PDU PDU_to_decode` – Snacc PDU structure used to hold decoded BER data received from target. After data is decoded into fields in this structure, the fields are mapped onto corressponding fields in a DBV PDU structure.

`PDU *PDU_ptr` – Pointer to `PDU_to_encode` structure used throughout the function and passed to subordinate functions.

`int return_value = SI_OK` – Used to store the current return value of the function.

`DbvProtocolStack eProtocolStack` – Stores the stack type which is used for the specified asssociation. Used to decide whether to wait for a PDU over the TCP or OSI transport.

`DBVAssociateDetails *psAssociateDetails` – Used to store the information about the specified association Id. The OSI/TCP specific association descriptor is accessed.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId;
DbvOriginData          *psOriginData = NULL;
DbvError               *psError = NULL;


if (DbvReceiveDataOrigin(lAssocId, CALL_BLOCKING, &psOriginData, &psError) == SI_OK)
{
        /* Process data */
        FreeOriginData(&psOriginData);
                  :
                  :
}
else
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                  :
                  :
}
                  :
                  :
```

8 TARGET FUNCTIONS

NOTE: While the Association functions are split into two distinct sections in this document - OSI and TCP services - the actual code is integrated into the same source file and functions. There is logic within the functions to determine whether to execute TCP or OSI service functionality.

8.1 Association Handling Functions Using the ACSE-Services

8.1.1 DbvTargetInitialize() Function

Identification:

```
int DbvTargetInitialize(      int           argc,
                              char        **argv,
                              DbvError    **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

This function parses the command line arguments given to it.

Functionality:

The function looks through the argv array and finds the information needed to correctly initialise the target system.

The function `AcInit()` is called to initialise ISODE. The command line parameters needed for ISODE initialisation are included automatically by the ISODE daemons.

There is only one argument which the API user must provide to the target: `"OSI"` or `"TCP"`. This is added automatically if using the `targetd` program but must be added after the program filename in the `inetd`, `iaed` or `tsapd` configuration files. See sections 15 and 16 of this document for complete details on configuring daemons

The global linked list for storing associations (only one in the case of a target) is also created (`DBVAssociateList`).

Subordinates:

The subordinates to this function are:

`SL_Create()`

`DBVAllocCheck()`

`DBVAlloc()`

`AcInit() - ISODE function`

`DBVLogISODEACSEError()`

`DBVfill_error_structure()`

Dependencies:

The dependencies for this function are:

- None - This is the first function that the target application should call.

Interfaces:

The function creates the linked list `DBVAssociateList` to which the unique id for this association will be added. The global variable `DBVOsiOrInternetAddress` is also updated with valid data.

`DBVpsAssocControlStart` passes information between this function and `ReceiveAssociateRequest()`.

Resources:

The function will use the following resources:

- OSI ISODE Unix libraries

References:

Additional information on this function can be found in:

Global variable information - section 11 of this document.

Internal function information - section 10 of this document.

ISODE Volume 10 Programmer's Guide: ASEs - Chapter 1.2.1 Association Establishment.

Processing:

None.

Data:

`int return_value = SI_OK` - Used throughout the function to store the current return value. *SI_OK* indicates success, *SI_NOTOK* failure.

`struct AcSAPindication *psAssocControlIndication = NULL;` - This structure is updated with diagnostic information if the call to `AcRelRequest()` fails.

`struct AcSAPabort *psAssocControlAbort = NULL;` - Set to point to a structure within the `psAssocControlIndication` structure.

`DBVAssociateDetails *psAssociateDetails` - Used to store the information about the specified association Id.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

main(int argc, char **argv)
{
        DbvError        *psError = NULL;
        DbvAssocId      lAssocId;

        if(DbvTargetInitialize(argc, argv, &psError)==SI_NOTOK)
        {
                /* Error in TargetInitialise() */
                        :
                        :
                PrivateErrorHandling();
        }

        /* DbvReceiveAssociateRequest(&lAssocId, etc..) */
}
```

8.1.2 DbvReceiveAssociateRequest() Function

Identification:

```
int DbvReceiveAssociateRequest(  DbvAssocId             *plAssocId,
                                 DbvSecs                 lSeconds,
                                 char                  **ppcOriginAddress,
                                 DbvApplicationProtocol  *peApplicationProtocol,
                                 DbvError              **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

Receive an associate request from the target.

Functionality:

Firstly a new association Id is created. As a target only ever has one association, the Id will always be 1.

Next, memory is allocated for the `DBVAssociateDetails` structure which will hold details internal to the API about the new association. The structure is then initialised.

Next, the function `DBVCheckApplicationContext()` is called to determine the application context specified by the origin. See section X.X for details of the logic used. The result is placed in `peApplicationProtocol` (one of: *SR_Protocol*, *Z3950_Protocol* or *Unidentified_Protocol*).

If the return value of the function is *SI_OK*, then the new association is added to the global linked list `DBVAssociateList` and the address of the origin is placed in `ppcOriginAddress` (using the ISODE function `paddr2str()`) .

If the return value of the function is *SI_NOTOK*, this may be because the ACSE configuration file, application context name or presentation context definition list is missing (DBV_ACSE_MISSING_PCDL_VALUE, DBV_ACSE_MISSING_ACN_VALUE or DBV_FILE) . If this is the case, the function will have completed as normal and the association will still be valid (indicated in `eAssociationState` of the error structure).

The value of `lSeconds` is effectively ignored as there are no blocking calls in this function.

Subordinates:

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`DBVfill_error_structure()`

`DBVCheckApplicationContext()`

`paddr2str()` – ISODE function

`SL_Add()`

Dependencies:

The dependencies for this function are:

- The machine must be connected to an OSI network.

Interfaces:

This function uses the `DBVAssociateList` global linked list variable. The function initialises the `DBVpsAssocControlStart` global variable with information from the call to ISODE function `AcInit()`.

Resources:

This function and its subordinates use the following resources:

- OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.1.1

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10 Programmer's Guide: ASEs - Chapter 1.2.1 Association Establishment (Address Encodings).

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` – Used to store the information about the specified association Id.

`static DbvAssocId IdGenerator = 0;` – Used to store the newly generated Id for the target.

`int return_value = SI_OK` – Used throughout the function to store the current return value. *SI_OK* indicates success, *SI_NOTOK* failure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

main(int argc, char **argv)
{
        DbvAssocId                      lAssocId;
        char                           *pcOriginAddress;
        DbvApplicationProtocol          eApplicationProtocol;
        DbvError                       *psError = NULL;
                        :
                /* A successful DbvTargetInitialize() must be done here */
                        :
        if(DbvReceiveAssociateRequest(&lAssocId, CALL_BLOCKING, &origin_address,
                &eApplicationProtocol, &psError) == SI_NOTOK)
```

```
{
        /* Error in DbvReceiveAssociateRequest */
        /* Look inside the error structure */
        if(psError->eErrorClass != SI_Abort)
        {
                if(psError->uErrorCode.lIfOsCoError == DBV_FILE ||
                        psError->uErrorCode.lIfOsCoError == DBV_ACSE_MISSING_PCDL_VALUE ||
                        psError->uErrorCode.lIfOsCoError == DBV_ACSE_MISSING_ACN_VALUE)
                {
                        /* An error concerning the ACSE config file! */

                }
                else
                {
                        /* Some other error (more serious!) */
                        PrivateErrorHandling(psError);
                }
        }
        else
        {
                /* Something serious has gone wrong! */
                PrivateErrorHandling(psError);
        }
}
        :
        :
etc....
```

8.1.3 DbvAssociateResponse() Function

Identification:

```
int DbvAssociateResponse(     DbvAssocId          lAssocId,
                              DbvAssociateRsp     eAssociateRsp,
                              DbvError          **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

Respond to an associate request using the service A-ASOCIATE.RESPONSE.

Function:

The function first checks that the association is valid and in the correct state.

If the value of `eAssociateRsp` is not *Accepted*, the function checks if the application protocol is *Unidentified_Protocol*. If so, the reason for rejection is set to *ACS_CONTEXT* (ISODE value). Otherwise the default reason is kept (*ACS_USER_NULL*).

The proposed presentation contexts are accepted or rejected a call to `DBVAcceptOrRejectPresentationContexts()`. Note that if the origin provides a presentation context definition list but the ACSE configuration file is missing or contains an error, the value of `eAssociateRsp` will be changed (internally) to *Rejected_Tran* and the association automatically rejected. The function will return *SI_NOTOK* and the error structure will indicate that the association no longer exists. If the origin does not provide a presentation context definition list, the function will accept or reject the association as indicated by the value of `eAssociateRsp`, but return with the appropriate error.

A call to the ISODE function `AcAssocResponse()` now accepts or rejects the association

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`SL_Current()`

`DBVAcceptOrRejectPresentationContexts()`

`AcAssocResponse()`

`DBVTrace()`

`DBVACSEOSICheckISODEError()`

`DBVLeaveFuncTrace()`

Dependencies:

The dependencies for this function are:

- The machine must be connected to an OSI network.

Interfaces:

This function uses the `DBVAssociateList` global linked list variable. The function uses the `DBVAssocControlStart` global variable to get the application context name and machine address of the origin.

Resources:

This function and its subordinates use the following resources:

- OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.1.2

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10 Programmer's Guide: ASEs - Chapter 1.2.1 Association Establishment.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`long lReason = ACS_USER_NULL;` – The Reason for rejection (if applicable). Defaults to no reason.

`long lAcceptOrReject = ACS_ACCEPT;` – The ISODE mapping for the eAssociateRsp service-user supplied variable.

`struct AcSAPindication *psAssocControlIndication=NULL;` – Filled out if the call to ISODE function `AcAssocResponse()` fails. Contains diagnostic information which is reported back to the service-user.

`struct AcSAPabort *psAssocControlAbort;` Points to the abort structure contained within the psAssocControlIndication structure.

`struct PSAPctxlist *psPresentationContextList=NULL;` – Set to point to the list which was supplied by the origin. The list is passed from the function DbvTargetInitialize() using the global variable DBVAssocControlStart.

`struct PSAPstart *psPresentationStart=NULL;` – Set to point to the PSAPstart structure contained within the DBVAssocControlStart global structure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

main(int argc, char **argv)
{
        DbvAssocId                      lAssocId;
        char                           *pcOriginAddress;
        DbvApplicationProtocol          eApplicationProtocol;
        DbvAssociateRsp                 eAssociateRsp = Accepted;
        DbvError                       *psError = NULL;
                        :
        /* A successful DbvTargetInitialize() must be done here */
                        :
        /* Now a DbvReceiveAssociateRequest() */
                        :
        /* For this example we only support SR */
        if(eApplicationProtocol!= SR_Protocol)
        {
                eAssociateRsp = Rejected_Tran;
        }

        if(DbvAssociateResponse(lAssocId, eAssociateRsp, &psError)==SI_NOTOK)
        {
                if(psError->eAssociationState != Association_Open)
                {
                        /* An error occurred and we can't continue */
                        PrivateErrorHandling(psError);
                                :
                                :
                }
                else
                {
                        /* There's been an error, but we're OK to continue */
                        LogTheError(psError);
                                :
                                :
                }
        }
        /* Now quit (if call rejected) or listen for incoming APDUs (DbvReceiveDataTarget() ) */
                        :
                        :
```

8.1.4 DbvReleaseResponse() Function

Identification:

```
int DbvReleaseResponse( DbvAssocId          lAssocId,
                        DbvReleaseRsp      *psReleaseRsp,
                        DbvError          **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

Respond to a release request. Send the A-RELEASE.RESPONSE action to the origin.

Function:

First the association is checked to make sure that it is valid. Next, it's state is checked. If the `psReleaseRsp` parameter is provided by the service-user as *(DbvReleaseRsp *)NULL* the function releases the association using reason *Normal*. If the parameter is completed, the function uses the supplied details to form a valid response. If the service-user tries to reject the release request, the function will return *SI_NOTOK*. This is because rejecting a release request is not permitted in the context of SR and Z39.50.

The release is completed with a call to the ISODE function `AcRelResponse()`.

If the release was successful and the association no longer exists, it is removed from the internal association list `DBVAssociateList`.

Subordinates:

`DBVEnterFuncTrace()`

`DBVCheckError()`

`DBVAllocCheck()`

`DBVAlloc()`

`SIWhichStack()`

`SL_Current()`

`DBVfill_error_structure()`

`AcRelResponse() - ISODE function`

`DBVTrace()`

`DBVACSEOSICheckISODEError()`

`SL_RemoveCurrent()`

`DBVLeaveFuncTrace()`

Dependencies:

The dependencies for this function are:

- The machine must be connected to an OSI network.
- The association must have received a `TA_ReleaseInd` from the origin.

Interfaces:

This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following resources:

- OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.1.3

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

ISODE Volume 10 Programmer's Guide: ASEs - Chapter 1.2.2 Association Release.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to hold the details stored internal to the API (service-provider specific details) concerning the specified association Id.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` – Used throughout the function to record the current return value of the function.

`int dStatus;` – Maps to the service-user supplied variable `psReleaseRsp->peReleaseRspReason`. Holds the ISODE equivalent of the DBV type.

`int dReason;` – Maps to the service-user supplied variable `psReleaseRsp->eReleaseRspResult`. Holds the ISODE equivalent of the DBV type.

`struct AcSAPindication *psAssocControlIndication=NULL;` – Filled out if the call to ISODE function `AcRelResponse()` fails. Contains diagnostic information which is reported back to the service-user.

`struct AcSAPabort *psAssocControlAbort;` Points to the abort structure contained within the psAssocControlIndication structure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

              :
              :
```

```
DbvAssocId                      lAssocId;
DbvTargetData                  *psTargetData;
DbvError                       *psError = NULL;
                :
.... DbvReceiveDataTarget(lAssocId, CALL_BLOCKING, &psTargetData, &psError) .....
                :
                :
switch (psTargetData->eTargetEvent)
{
         case TA_InitializeInd:
                :
                :
case TA_ReleaseInd:
        if(DbvReleaseResponse(lAssocId, (DbvReleaseRsp *)NULL, &psError) == SI_NOTOK)
        {
                /* The release was not successful */
                PrivateErrorHandling(psError);
        }
        else
        {
                /* The Release was successful */
        }
        break;
case TA_Invalid_APDU:
        print_target_invalidapdu(psTargetData);
default:
        break;
 }
```

8.1.5 DbvAbortRequest() Function

Identification:

```
int DbvAbortRequest(    DbvAssocId          lAssocId,
                        DbvError          **ppsError );
```

Refer to 7.1.3

8.2 Association Handling Functions Using TCP Services

8.2.1 DbvTargetInitialize() Function

Identification:

```
int DbvTargetInitialize(      int           argc,
                              char        **argv,
                              DbvError    **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

This function parses the command line arguments given to it.

Functionality:

Refer to section 8.1.1.

Subordinates:

The subordinates to this function are:

`SL_Create()` - Creates the DBVAssociateList linked list

`DBVCheckError()`

Dependencies:

The dependencies for this function are:

- None - This is the first function that the target application should call.

Interfaces:

The function creates the linked list `DBVAssociateList` to which the unique Id and details for this association will be added. The global variables `DBVsocket`, `DBVaddress` and `DBVOsiOrInternetAddress` are also updated with valid data.

Resources:

The function will use the following resources:

- None

References:

Additional information on this function can be found in:

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

None

Data:

`int return_value = SI_OK` – Used throughout the function to store the current return value.

Usage:

```
main(int argc, char *argv[])
{
        DbvAssocId      lAssocId;
        DbvError       *psError = NULL;
                :
                :
        if(DbvTargetInitialize(argc, argv, &psError)==SI_NOTOK)
        {
                /* Error in TargetInitialise() */
                        :
                        :
                PrivateErrorHandling();
        }

        /* DbvReceiveAssociateRequest(&lAssocId, etc..) */

                :
```

8.2.2 DbvReceiveAssociateRequest() Function

Identification:

The identification of the function is:

```
int DbvReceiveAssociateRequest( DbvAssocId             *plAssocId,
                                DbvSecs                 lSeconds,
                                char                  **ppcOriginAddress,
                                DbvApplicationProtocol  *peApplicationProtocol,
                                DbvSIError            **ppsSIError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

Assigns a unique identifier to the association. Returns the address of the remote connection and the application protocol supported.

Functionality:

The function assigns an association Id to the `lAssocId` variable. `peApplicationProtocol` is set to *Unknown_protocol* and the `ppcOriginAddress` variable is filled with the internet address of the association originator, in standard dot notation (i.e. 132.45.65.7).

Details specific to the new association are added to the `psAssociateDetails` structure.

If the return value of the function is *SI_OK*, then the new association details structure `psAssociateDetails` is added to the global association linked list `DBVAssociateList`.

Subordinates:

`DBVCheckError()`

`SL_Locate()`

`DBVfill_error_structure()`

`SL_Add()`

Dependencies:

The dependencies of this function are:

- The function `DbvTargetInitialize()` must have already been called.

Interfaces:

This function will add a new `AssociateDetails` structure to the global linked list `DBVAssociateList`. This function uses the variables `DBVOsiOrInternetAddress`, `DBVsocket` and `DBVaddress` which were initialised by the `DbvTargetInitialize()` function.

Resources:

This function will use the following resources:

- Unix TCP/IP libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.2.1

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to build and store details about the new association. Although only one association will ever exist at the target, the details are placed in a linked list to maintain compatibility with the utility and various internal functions.

`int return_value = SI_OK` - Used to record the return value of the function.

`static AssocId IdGenerator = 0` - This function is used to generate the unique association id. It is also used to check that this function is not called more than once.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                      lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
char                           *pcOriginAddress = NULL;
DbvApplicationProtocol          eApplicationProtocol = Unidentified_Protocol;
DbvError                       *psError = NULL;

                :
        /* Do DbvTargetInitialize() first */
                :
        if (DbvReceiveAssociateRequest(&lAssocId, CALL_BLOCKING, &pcOriginAddress,
                &eApplicationProtocol, &psError) == SI_OK)
        {
         /*
                inspect pcOriginAddress and eApplicationProtocol
                if values acceptable, accept, else reject association
        */
                DbvAssociateResponse(lAssocId, .... , &psError);
                        :
                        :
        }
        else
        {
                PrivateErrorHandling(psError);
                        :
                        :
        }

        FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
                :
```

8.2.3 DbvAssociateResponse() Function

Identification:

```
int DbvAssociateResponse( DbvAssocId         lAssocId,
                          DbvAssociateRsp    eAssociateRsp,
                          DbvError         **ppsError );
```

Type:

This function is in source code file **targetassoc.c** which itself is contained in the library **libdbvtarget**.

Purpose:

Decide whether or not to serve the requests of a particular origin system.

Functionality:

The function first checks that the association supplied is a valid one. Next, if the value of `eAssociateRsp` is *Accepted* then the function instantly returns *SI_OK*. If the value is *Rejected_Permanent* or *Rejected_Transient*, then the socket is closed and the association removed from the list.

Subordinates:

`SL_Locate()`

`SL_RemoveCurrent()`

`DBVfill_error_structure()`

Dependencies:

The dependencies for this function are:

- The association should exist

Interfaces:

The function will remove a `DBVAssociateDetails` structure from the global linked list `DBVAssociateList`.

Resources:

This function will use the following resources:

- Unix TCP/IP libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.2.2

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` - Used to make sure that the association exsists before closing the connection.

`DbvProtocolStack eProtocolStack` - Used to store the protocol stack used by the association.

`int return_value = SI_OK` - Used to record the return value of the function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId;
DbvAssociateRsp         eAssociateRsp = Accepted;
DbvError               *psError = NULL;

        /* Do DbvTargetInitialize() and DbvReceiveAssociateRequest() first */

        if (DbvAssociateResponse(lAssocId, eAssociateRsp, &psError) == SI_OK)
        {
           /* association established */

           /* e.g. wait for incoming evets */
        }
        else
        {
           /* if psError->eAssociationState == Association_Open
              the association had been established even though an
              error occurred!!
           */

        PrivateErrorHandling(psError);
        FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
                  :
                  :
```

8.2.4 DbvReleaseResponse() Function

Identification:

```
int DbvReleaseResponse( DbvAssocId               lAssocId,
                        DbvReleaseRsp           *psReleaseRsp,
                        DbvSIError             **ppsSIError );
```

Type:

The function is in source code file **targetassoc.c** which is contained in the library **libdbvtarget**.

Purpose:

The purpose of this function is to release the transport connection using the TCP service `close()`.

Functionality:

The function checks to which communication stack the association belongs. In the case of a TCP connection, the socket corresponding to the association is closed.

The parameter `psReleaseRsp` may either be completed or specified as `(DbvReleaseRsp *)NULL`.

To maintain consistency between OSI and TCP, the function will return *SI_NOTOK* if the service-user tries to reject the release request.

Subordinates:

The subordinates to this function are:

`DBVEnterFuncTrace()`

`DBVLeaveFuncTrace()`

`SIWhichStack()`

`DBVfill_error_structure()`

`SL_RemoveCurrent()`

Dependencies:

The dependencies for this function are:

- The association should exist
- The association must have received a `TA_ReleaseInd` from the origin.

Interfaces:

The function will remove an `AssociateDetails` structure from the global linked list `DBVAssociateList`.

Resources:

This function will use the following resources:

- None

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.2.2.3

Global variable information - section 11.0 of this document.

Internal function information - section 10.0 of this document.

Processing:

None

Data:

`DBVAssociateDetails *psAssociateDetails` – Used to make sure that the association exsists before closing the connection.

`int return_value = SI_OK` – Used to record the return value of the function.

`ProtocolStack eProtocolStack` – Used to hold the stack type of the association.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

               :
               :
        DbvAssocId                      lAssocId;
        DbvTargetData                  *psTargetData;
        DbvError                       *psError = NULL;
                       :
        .... DbvReceiveDataTarget(lAssocId, CALL_BLOCKING, &psTargetData, &psError) .....
                       :
                       :
        switch (psTargetData->eTargetEvent)
        {
                 case TA_InitializeInd:
                       :
                       :
        case TA_ReleaseInd:
                if(DbvReleaseResponse(lAssocId, (DbvReleaseRsp *)NULL, &psError) == SI_NOTOK)
                {
                        /* The release was not successful */
                        PrivateErrorHandling(psError);
                }
                else
                {
                        /* The Release was successful */
                }
                break;
        case TA_Invalid_APDU:
                print_target_invalidapdu(psTargetData);
        default:
                break;
         }
```

8.2.5 DbvAbortRequest() Function

Identification:

```
int DbvAbortRequest(    DbvAssocId          lAssocId,
                        DbvSIError        **ppsSIError );
```

See Section 7.2.3

8.4 SR-/Z39.50 Functions

8.4.1 DbvInitializeResponse() Function

Identification:

```
int DbvInitializeResponse(    DbvAssocId          lAssocId,
                              DbvInitializeRsp   *psInitializeRsp,
                              DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_init.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 InitializeResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psInitializeRsp structure are checked by calling subordinate function DBVCheckDbvInitRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows an InitializeResponse PDU. If sending an InitializeResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psInitializeRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_initresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding

uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *SI_OK* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.1

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvInitializeResponse(DbvAssocId lAssocId, DbvInitializeRsp
*psInitializeRsp, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
```

```
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVInitializeResponse", lAssocId);

      return_value =  FALSE;

      /* If fields in psInitializeRsp are valid */
      IF (DBV_CHECK_DBV_INITRSP_FIELDS(lAssocId, psInitializeRsp, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Init Response is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_initresponse_structure;

            /* Map fields in psInitializeRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_INITRESPONSE(PDU_ptr, psInitializeRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvInitializeResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc InitializeResponse PDU structure fields (DBVsetup_snacc_intresponse()) and Snacc InitializeResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psInitializeRsp fields, the association and the state machine state are all valid for an InitializeResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psInitializeRsp fields onto Snacc InitializeResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc InitializeResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvInitializeRsp       *psInitializeRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psInitializeRsp and fill in structure.
*/

if (DbvInitializeResponse(lAssocId, psInitializeRsp, &psError) == SI_OK)
{
   FreeInitializeRsp(&psInitializeRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeInitializeRsp(&psInitializeRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.2 DbvSearchResponse() Functions

Identification:

```
int DbvSearchResponse(  DbvAssocId          lAssocId,
                        DbvSearchRsp       *psSearchRsp,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_search.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 SearchResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psSearchRsp structure are checked by calling subordinate function DBVCheckDbvSearchRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a SearchResponse PDU. If sending a SearchResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psSearchRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_srchresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.1.3.2

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvSearchResponse(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVSearchResponse", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psSearchRsp are valid */
      IF (DBV_CHECK_DBV_SRCHRSP_FIELDS(lAssocId, psSearchRsp, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Search Response is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_srchresponse_structure;

            /* Map fields in psSearchRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_SRCHRESPONSE(PDU_ptr, psSearchRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvSearchResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc SearchResponse PDU structure fields (DBVsetup_snacc_srchresponse()) and Snacc SearchResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psSearchRsp fields, the association and the state machine state are all valid for a SearchResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psSearchRsp fields onto Snacc SearchResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc SearchResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSearchRsp           *psSearchRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psSearchRsp and fill in structure.
*/

if (DbvSearchResponse(lAssocId, psSearchRsp, &psError) == SI_OK)
{
   FreeSearchRsp(&psSearchRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeSearchRsp(&psSearchRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.3 DbvPresentResponse() Functions

Identification:

```
int DbvPresentResponse( DbvAssocId          lAssocId,
                        DbvPresentRsp      *psPresentRsp,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_present.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 PresentResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psPresentRsp structure are checked by calling subordinate function DBVCheckDbvPresRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a PresentResponse PDU. If sending a PresentResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psPresentRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_presresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.3

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvPresentResponse(DbvAssocId lAssocId, DbvPresentRsp
*psPresentRsp, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVPresentResponse", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psPresentRsp are valid */
      IF (DBV_CHECK_DBV_PRESRSP_FIELDS(lAssocId, psPresentRsp, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Present Response is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_presresponse_structure;

            /* Map fields in psPresentRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_PRESRESPONSE(PDU_ptr, psPresentRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvPresentResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc PresentResponse PDU structure fields (DBVsetup_snacc_presresponse()) and Snacc PresentResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psPresentRsp fields, the association and the state machine state are all valid for a PresentResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psPresentRsp fields onto Snacc PresentResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc PresentResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvPresentRsp          *psPresentRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psPresentRsp and fill in structure.
*/

if (DbvPresentResponse(lAssocId, psPresentRsp, &psError) == SI_OK)
{
   FreePresentRsp(&psPresentRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreePresentRsp(&psPresentRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.4 DbvDeleteResultSetResponse() Functions

Identification:

```
int DbvDeleteResultSetResponse(     DbvAssocId          lAssocId,
                                    DbvDeleteResultSetRsp
*psDeleteResultSetRsp,
                                    DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_delete.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 DeleteResultSetResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psDeleteResultSetRsp structure are checked by calling subordinate function DBVCheckDbvDelRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a DeleteResultSetResponse PDU. If sending a DeleteResultSetResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psDeleteResultSetRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_delresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.4

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvDeleteResultSetResponse(DbvAssocId lAssocId,
DbvDeleteResultSetRsp *psDeleteResultSetRsp, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVDeleteResultSetResponse", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psDeleteResultSetRsp are valid */
      IF (DBV_CHECK_DBV_DELRSP_FIELDS(lAssocId, psDeleteResultSetRsp, ppsError)
THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the DeleteResultSet Response is valid and can be sent to a valid
associated origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_delresponse_structure;

            /* Map fields in psDeleteResultSetRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_DELRESPONSE(PDU_ptr, psDeleteResultSetRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvDeleteResultSetResponse", lAssocId,
return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc DeleteResultSetResponse PDU structure fields (DBVsetup_snacc_delresponse()) and Snacc DeleteResultSetResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either

OSI_Stack or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psDeleteResultSetRsp fields, the association and the state machine state are all valid for a DeleteResultSetResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psDeleteResultSetRsp fields onto Snacc DeleteResultSetResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc DeleteResultSetResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvDeleteResultSetRsp         *psDeleteResultSetRsp = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psDeleteResultSetRsp and fill in structure.
*/

if (DbvDeleteResultSetResponse(lAssocId, psDeleteResultSetRsp, &psError) == SI_OK)
{
   FreeDeleteResultSetRsp(&psDeleteResultSetRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeDeleteResultSetRsp(&psDeleteResultSetRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.5 DbvScanResponse() Functions

Identification:

```
int DbvScanResponse(    DbvAssocId          lAssocId,
                        DbvScanRsp         *psScanRsp,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_scan.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 ScanResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psScanRsp structure are checked by calling subordinate function DBVCheckDbvScanRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a ScanResponse PDU. If sending a ScanResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psScanRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_scanresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *SI_OK* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.5

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvScanResponse(DbvAssocId lAssocId, DbvScanRsp *psScanRsp,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;
```

```
      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVScanResponse", lAssocId);

      return_value =  FALSE;

      /* If fields in psScanRsp are valid */
      IF (DBV_CHECK_DBV_SCANRSP_FIELDS(lAssocId, psScanRsp, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Scan Response is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_scanresponse_structure;

            /* Map fields in psScanRsp structure onto fields in SNACC structure
*/
            DBV_SETUP_SNACC_SCANRESPONSE(PDU_ptr, psScanRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvScanResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ScanResponse PDU structure fields (DBVsetup_snacc_scanresponse()) and Snacc ScanResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psScanRsp fields, the association and the state machine state are all valid for a ScanResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psScanRsp fields onto Snacc ScanResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ScanResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvScanRsp             *psScanRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psScanRsp and fill in structure.
*/

if (DbvScanResponse(lAssocId, psScanRsp, &psError) == SI_OK)
{
   FreeScanRsp(&psScanRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeScanRsp(&psScanRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.6 DbvResourceReportResponse() Functions

Identification:

```
int DbvResourceReportResponse(DbvAssocId          lAssocId,
                              DbvResourceReportRsp
*psResourceReportRsp,
                              DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_resrep.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 ResourceReportResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psResourceReportRsp structure are checked by calling subordinate function DBVCheckDbvResRepRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a ResourceReportResponse PDU. If sending a ResourceReportResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psResourceReportRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_resrepresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.6

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvResourceReportResponse(DbvAssocId lAssocId,
DbvResourceReportRsp *psResourceReportRsp, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVResourceReportResponse", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psResourceReportRsp are valid */
      IF (DBV_CHECK_DBV_RESREPRSP_FIELDS(lAssocId, psResourceReportRsp,
ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the ResourceReport Response is valid and can be sent to a valid
associated origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_resrepresponse_structure;

            /* Map fields in psResourceReportRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_RESREPRESPONSE(PDU_ptr, psResourceReportRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvResourceReportResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ResourceReportResponse PDU structure fields (DBVsetup_snacc_resrepresponse()) and Snacc ResourceReportSetResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either

OSI_Stack or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psResourceReportRsp fields, the association and the state machine state are all valid for a ResourceReportResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psResourceReportRsp fields onto Snacc ResourceReportResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ResourceReportResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvResourceReportRsp          *psResourceReportRsp = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psResourceReportRsp and fill in structure.
*/

if (DbvResourceReportResponse(lAssocId, psResourceReportRsp, &psError) == SI_OK)
{
   FreeResourceReportRsp(&psResourceReportRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeResourceReportRsp(&psResourceReportRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.7 DbvSortResponse() Functions

Identification:

```
int DbvSortResponse(    DbvAssocId          lAssocId,
                        DbvSortRsp         *psSortRsp,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_sort.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 SortResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psSortRsp structure are checked by calling subordinate function DBVCheckDbvSortRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a SortResponse PDU. If sending a SortResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psSortRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_sortresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.7

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvSortResponse(DbvAssocId lAssocId, DbvSortRsp *psSortRsp,
DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVSortResponse", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psSortRsp are valid */
      IF (DBV_CHECK_DBV_SORTRSP_FIELDS(lAssocId, psSortRsp, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Sort Response is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_sortresponse_structure;

            /* Map fields in psSortRsp structure onto fields in SNACC structure
*/
            DBV_SETUP_SNACC_SORTRESPONSE(PDU_ptr, psSortRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvSortResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc SortResponse PDU structure fields (DBVsetup_snacc_sortresponse()) and Snacc SortResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psSortRsp fields, the association and the state machine state are all valid for a SortResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psSortRsp fields onto Snacc SortResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc SortResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSortRsp             *psSortRsp = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psSortRsp and fill in structure.
*/

if (DbvSortResponse(lAssocId, psSortRsp, &psError) == SI_OK)
{
   FreeSortRsp(&psSortRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeSortRsp(&psSortRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.8 DbvSegmentationRequest() Functions

Identification:

```
int DbvSegmentRequest(  DbvAssocId          lAssocId,
                        DbvSegmentReq      *psSegmentReq,
                        DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_segment.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 SegmentResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psSegmentReq structure are checked by calling subordinate function DBVCheckDbvSegReqFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows a SegmentRequest PDU. If sending a SegmentRequest is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psSegmentReq structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_segrequest(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *`SI_OK`* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.8

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvSegmentRequest(DbvAssocId lAssocId, DbvSegmentReq
*psSegmentReq, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;

      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVSegmentRequest", lAssocId);

      return_value =  FALSE;
```

```
      /* If fields in psSegmentReq are valid */
      IF (DBV_CHECK_DBV_SEGREQ_FIELDS(lAssocId, psSegmentReq, ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the Segment Request is valid and can be sent to a valid associated
origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_segmentrequest_structure;

            /* Map fields in psSegmentRsp structure onto fields in SNACC
structure */
            DBV_SETUP_SNACC_SEGREQ(PDU_ptr, psSegmentRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvSegmentResponse", lAssocId, return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file **z39_50_apdu_1994.h**. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc SegmentRequest PDU structure fields (DBVsetup_snacc_segrequest()) and Snacc SegmentRequest PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psSegmentReq fields, the association and the state machine state are all valid for a SegmentRequest PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psSegmentReq fields onto Snacc SegmentRequest structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc SegmentRequest PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvSegmentReq          *psSegmentReq = NULL;
DbvError               *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psSegmentReq and fill in structure.
*/

if (DbvSegmentRequest(lAssocId, psSegmentReq, &psError) == SI_OK)
{
   FreeSegmentReq(&psSegmentReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeSegmentReq(&psSegmentReq);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.9 DbvExtendedServicesResponse() Functions

Identification:

```
int DbvExtendedServicesResponse(    DbvAssocId          lAssocId,
                                    DbvExtendedServicesRsp
*psExtendedServicesRsp,
                                    DbvError          **ppsError);
```

Type:

This function is in source code file **target_z3950_extend.c** which itself is contained in the library **libdbtarget**

Purpose:

The purpose of this function is to send a Z39.50 ExtendedServicesResponse PDU to the origin of the current association.

Function:

a) Function tracing is enabled.

b) The variable return_value is set to `SI_NOTOK`.

c) Check 1: The fields in the psExtendedServicesRsp structure are checked by calling subordinate function DBVCheckDbvExtendRspFields() to ensure that they contain valid entries. If an invalid field is detected, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If all the fields are found to be valid the subordinate function returns with `SI_OK`.

d) Check 2: If check 1 returns `SI_OK`, the `lAssocId` association is checked by calling subordinate function SIWhichStack() to ensure that it is valid. If the association is invalid, the `ppsError` structure is allocated, completed with diagnostic information and the subordinate function returns with `SI_NOTOK`. If the association is valid the subordinate function returns with `SI_OK`.

e) Check 3: If check 2 returns `SI_OK`, the internal state machine is checked to ensure that the current state of the association allows an ExtendedServicesResponse PDU. If sending an ExtendedServicesResponse is invalid for the current state, the `ppsError` structure is allocated and completed with diagnostic information. If the operation is valid for the current state the variable return_value is set to `SI_OK`.

f) If the variable return_value has been set to `SI_OK` by check 1, check 2 then check 3 succeeding, the fields in the psExtendedServicesRsp structure are mapped onto similar fields in a related 'snacc structure' defined in snacc generated file **z39_50_apdu_1994.h** by calling subordinate function DBVsetup_snacc_extendresponse(). Snacc structures and their usage are further explained in section 'SR-/Z39.50 Subordinate Functions' below.

g) The completed snacc structure is 'encoded' from its structure form into a sequence of bytes then sent to the target by calling subordinate function DBVencode_and_send(). The encoding uses functions in snacc generated file **z39_50_apdu_1994.c** and snacc run-time library functions.

h) Function tracing is diabled.

i) The value *SI_OK* is returned.

Subordinates:

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of this function are:

- The association to the origin must be valid.

This PDU can only be sent if the internal state machine is in an appropriate state.

See section 5 for the Z39.50 PDU/Association state machine used within DBV-OSI II.

Interfaces:

This function effects the following global variables:

- This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its sub-ordintates use the following extra system resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

Document D-008 API-Definition, section 3.2.3.9

Global variable information - section 10.0 of this document.

Internal function information - section 9.0 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

The structure of the function is reproduced below in Structured English:

```
BEGIN FUNCTION DbvExtendedServicesResponse(DbvAssocId lAssocId,
DbvExtendedServicesRsp *psExtendedServicesRsp, DbvError **ppsSIError):

      DbvProtocolStack &eProtocolStack;
      int return_value = FALSE;
      SNACC_PDU *PDU_ptr = NULL;
```

```
      /* Start function tracing */
      DBV_ENTER_FUNC_TRACE("DBVExtendedServicesResponse", lAssocId);

      return_value =  FALSE;

      /* If fields in psExtendedServicesRsp are valid */
      IF (DBV_CHECK_DBV_EXTENDRSP_FIELDS(lAssocId, psExtendedServicesRsp,
ppsError) THEN

            /* If valid communication stack exists */
            IF (SIWhichStack(lAssocId, &eProtocolStack, ppsSIError)) THEN

                  /* If in valid state machine state */
                  IF (state_machine_for_current_association == SI_OK) THEN
                        return_value = TRUE;
                  ENDIF
            ENDIF
      ENDIF

      /* If the ExtendedServices Response is valid and can be sent to a valid
associated origin */
      IF (return_value == TRUE) THEN

            /* Allocate space for snacc structure */
            PDU_ptr = malloc_snacc_extendresponse_structure;

            /* Map fields in psExtendedServicesRsp structure onto fields in
SNACC structure */
            DBV_SETUP_SNACC_EXTENDRSP(PDU_ptr, psExtendedServicesRsp);

            /* Encode SNACC structure in 'BER' data stream and send it to target
*/
            DBV_ENCODE_AND_SEND(PDU_ptr, lAssocId, ppsSIError);
      ENDIF

      /* Leave function tracing */
      DBV_LEAVE_FUNC_TRACE("DbvExtendedServicesResponse", lAssocId,
return_value);
      return return_value;

END FUNCTION.
```

Data:

`PDU pdu_to_encode` - Snacc PDU structure defined in snacc generated file ***z39_50_apdu_1994.h***. This structure contains a huge tree of sub structures corressponding to all fields in all PDUS.

`PDU *PDU_ptr` - Pointer to a Snacc PDU structure. This pointer is set to the address of structure `pdu_to_encode` then passed when appropriate to subordinate functions which control mapping of Dbv->Snacc ExtendedServicesResponse PDU structure fields (DBVsetup_snacc_extendresponse()) and Snacc ExtendedServicesResponse PDU encoding and sending (DBVencode_and_send()).

`DBVAssociateDetails *psAssociateDetails` - Pointer to an association structure containing information about the current `lAssocId` association. This pointer is set to the address of the association structure held in the global variable `DBVAssociateList`.

`DbvProtocolStack eProtocolStack` - Enumeration (integer) variable set to the type of communication stack being used by the current association if the current association is valid. This integer, which is set by calling subordinate function SIWhichStack(), will be set to either *OSI_Stack* or *TCP_Stack* if the association is valid. If the association is invalid when it is called SIWhichStack() returns `SI_NOTOK`.

`int return_value` - Integer variable used to indicate the legality of the PDU operation. This variable is set at the start of the function to value *SI_NOTOK*. If the psExtendedServicesRsp fields, the association and the state machine state are all valid for an ExtendedServicesResponse PDU send, the value of the variable is changed to *SI_OK*. Further subordinate functions to map the psExtendedServicesRsp fields onto Snacc ExtendedServicesResponse structure fields then encode and send the Snacc PDU to the association target are only called if `return_value` is changed to *SI_OK*.

`int val` - Integer variable used to hold 0 returned from a setjmp() macro call made in the function then -1 returned by any subsequent longjmp() calls made by Dbv->Snacc PDU field mapping subordinate functions. A longjmp(-1) call, which can be made from various points within the subordinate mapping functions, is made if any memory error is detected (any alloc returns NULL) when allocating space for Snacc ExtendedServicesResponse PDU fields. If a longjmp() occurs, the DBV error structure is completed with diagnostic information.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId                    lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvExtendedServicesRsp        *psExtendedServicesRsp = NULL;
DbvError                      *psError = NULL;

/*
    Suppose the call to function DbvAssociateRequest()
    had been successful and lAssocId is a valid identifier.

    Allocate buffer for psExtendedServicesRsp and fill in structure.
*/

if (DbvExtendedServicesResponse(lAssocId, psExtendedServicesRsp, &psError) == SI_OK)
{
   FreeExtendedServicesRsp(&psExtendedServicesRsp);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeExtendedServicesRsp(&psExtendedServicesRsp);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                :
                :
}
                :
                :
```

8.4.10 DbvCloseRequest() Functions

See the DbvCloseRequest() description under the 'Functions Used By the Origin' section.

8.4.11 DbvCloseResponse() Functions

See the DbvCloseResponse() description under the 'Functions Used By the Origin' section.

8.4.12 DbvReceiveDataTarget() Functions

Identification:

```
int DbvReceiveDataTarget(     DbvAssocId              lAssocId,
                              DbvSecs                 lSeconds,
                              DbvTargetData         **ppsTargetData,
                              DbvSIError            **ppsSIError )
```

Type:

This function is in source code file **target_Z3950_main.c** and is included in the library **libdbvtarget**.

Purpose:

Waits for a complete PDU of Z39.50 data to be sent by the origin, decodes the data then sets ppsTargetData appropriately to contain the PDU structure format decoded from the data.

Functionality:

a) Function tracing is invoked by calling DBVEnterFuncTrace().

b) The association is checked to ensure that is it valid by calling SIWhichStack(). If the association is valid, variable `return_value` is set to *SI_OK*, if it is invalid `return_value` is set to *SI_NOTOK*.

c) If the assocition is valid (`return_value` set to *SI_OK*) then a complete PDU of data is received from the origin and decoded into a Snacc PDU structure by calling either

c1) DBVreceive_data_over_socket() if TCP/IP communication is being used by the current association

c2) DBVreveive_data_over_OSI() if OSI communication is being used by the current association

If an error occurs during data reception then the receive function being used returns with *SI_NOTOK*.

d) If data for a complete Snacc PDU has been received and decoded by DBVreceive_data_over_socket()/DBVreceive_data_over_OSI(), function DBVconvert_PDU_to_target_data() is called to map the Snacc PDU data onto structures in the ppsTargetData structure (see **Section 10.10** - PDU Checking And Mapping Subordinate Functions). Any field allocing errors in DBVconvert_PDU_to_target_data() or any of its subordinates are caught and trapped by a setjmp(), lonjmp() mechanism and result in the error structure passed as a parameter to the function being completed with diagnostic information. Variable `return_value` is set to *SI_OK* if all mapping succeeds or *SI_NOTOK* if mapping fails.

e) Function tracing is turned off.

Subordinates:

The subordinates to this function are:

`DBVEnterFuncTrace()`

`DBVLeaveFuncTrace()`

`SL_Current()`

`SIWhichStack()`

`DBVreceive_data_over_socket()`

`DBVreceive_data_over_osi()`

`DBVconvert_pdu_to_target_data()`

SIL developed functions called by this function are fully detailed in section 'PDU Checking And Mapping Subordinate Functions'.

Dependencies:

The dependencies of thi function are:

- The association should be valid
- The Z39.50 state machine must be satisfied.

Interfaces:

This function uses the `DBVAssociateList` global linked list variable.

Resources:

This function and its subordinates use the following resources:

- TCP/IP and OSI (ISODE) Unix libraries.

References:

Additional information on this function can be found in:

D-008 API-Definition, Section 3.2.3.12

Global variable information - Section 11 of this document.

Internal function information - Section 10 of this document.

Z39.50 internal state machine - Section 5 of this document.

Section 'PDU Checking And Mapping Subordinate Functions' in this document.

Processing:

```
FUNCTION DbvReceiveDataTarget( DbvAssocId lAssocId, DbvSecs lSeconds,
DbvTargetData **ppsTargetData, DbvError **ppsError):


      PDU PDU_to_decode;
```

```
      PDU *PDU_ptr;
      int return_value = SI_OK;
      DbvProtocolStack eProtocolStack;

      PDU_ptr = &PDU_to_decode;

      DBV_ENTER_FUNC_TRACE("ReceiveDataTarget",lAssocId);
      IF  (SIWhichStack(lAssocId,  &eProtocolStack,  ppsError)  ==  SI_OK)  THEN

            eProtocolVersion = STACK_OF_CURRENT_ASSOCIATION();
            IF (eProtocolVersion == TCP) THEN
                        return_value                                          =
DBV_RECEIVE_DATA_OVER_TCP_AND_CONVERT_INTO_SNACCPDU(lAssocId, PDU_ptr);
            ELSE
                  return_value                                                =
DBV_RECEIVE_DATA_OVER_OSI_AND_DECODE_INTO_SNACCPDU(lAssocId, PDU_ptr);
            ENDIF
      ENDIF

      IF (return_value == SI_OK) THEN
            return_value  =  DBV_CONVERT_SNACCPDU_TO_ORIGIN_DATA(*ppsTargetData,
PDU_ptr);
      ENDIF

      DBV_LEAVE_FUNC_TRACE("ReceiveDataTarget",lAssocId, return_value);
      return return_value;
END FUNCTION.
```

Data:

`PDU PDU_to_decode` - Snacc PDU structure used to hold decoded BER data received from origin. After data is decoded into fields in this structure, the fields are mapped onto corressponding fields in a DBV PDU structure.

`PDU *PDU_ptr` - Pointer to `PDU_to_decode` structure used throughout the function and passed to subordinate functions.

`ProtocolStack eProtocolStack` - Used to hold the protocol used by the association (*`TCP_Stack`* or *`OSI_Stack`*).

`DBVAssociateDetails *psAssociateDetails` - Holds the details of the specified association. The OSI/TCP specific association descriptor is accessed.

`int return_value = SI_OK` - Stores the current return value of the function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId;
DbvTargetData          *psTargetData = NULL;
DbvError               *psError = NULL;


if (DbvReceiveDataTarget(lAssocId, CALL_BLOCKING, &psTargetData, &psError) == SI_OK)
{
        /* Process data */
        FreeTargetData(&psTargetData);
                  :
                  :
}
else
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);   /* release buffer, assign NULL pointer */
                  :
                  :
}
                  :
                  :
```

9 UTILITY FUNCTIONS

9.1 SIGetEventLocation() Function

Identification:

```
int SIGetEventLocation( DbvAssocId        **pplAssocId,
                        DbvSecs             lSeconds,
                        DbvFds            **ppulFileDescriptor,
                        DbvEventLocation  **ppsEventLocation,
                        DbvSIError        **ppsSIError );
```

Type:

This function is in file **si_utilities.c** and is contained in the library **libdbv**.

Purpose:

Notify the API-User if there are any data to be received.

Functionality:

This function takes NULL terminated lists of association Ids and file descriptors and waits for incoming data for the time specified in lSeconds.

Firstly, the lists are checked to make sure that they are valid. If they pass this check, then the lists are checked to make sure that there is a least one item to wait on.

Next, the Association Id's are each checked for validity. For TCP/IP associations the unix macro `FD_SET()` is used to place the file descriptor in a list so that a `select()` call can be performed later. The ISODE function `PSelectMask()` is used to perform the equivalent service for OSI associations.

The service-user file descriptors held in the `ppulFileDescriptor` list are then also placed into a list using the Unix macro `FD_SET()`.

Now a `while()` loop is executed which checks the file descriptors/TCP associations and OSI associations in turn. This continues until an error occurs, a timeout occurs or incoming data is detected.

If incoming data is detected then the internal function `DBVfill_out_event_location_structure()` is called to fill out the `ppsEventLocation` structure with the associations and file descriptors which are ready for reading (see Section 10.9.1).

Subordinates:

The subordinates to this function are:

`DBVfill_error_structure()`

`SIWhichStack()`

```
void DBVfill_out_event_location_structure()
```

Dependencies:

The dependencies of this function are:

- The machine must be connected to a TCP/IP network (if TCP/IP associations/file descriptors are present in the lists).
- The machine must be connected to an OSI compatible network (if OSI associations are present in the list).
- The Association Id's should be valid.
- The file descriptors should be valid.

Interfaces:

None

Resources:

This function will use the following resources:

- TCP/IP and/or OSI (ISODE) Unix libraries

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.3.1

The Network programming guide for your flavour of Unix.

The ISO Development Environment: User's Manual. Volume 1: Application Services & Volume 2: Underlying Services.

Processing:

None.

Data:

`int count,` - Used to index the arrays of associations and file descriptors.

`seconds,` – Holds the number seconds to listen for activity.

`time_up,` – A Flag which is set to DBV_TRUE if the function has listened for enough time.

`incoming;` – Set to true if either `select()` (TCP) or `xselect()` (OSI) indicate activity on an association/file descriptor.

`int select_result, osi_select_result;` - Hold the result of a TCP `select()` or OSI `xselect()` call respecitvely.

`int return_value = SI_OK;` - Used throughout the function to record the current return value.

`struct timeval time_out;` - Stores the length of time that `select()` will listen before returning.

`time_t start_time,` – Stores the system time when the listens were begun.

`current_time;` – Stores the current system time.

`fd_set read_ready,` - The list of TCP associations and file descriptors and their status (whether any data is waiting).

`set_read_ready,` - The initial list with all entries set to their initial values.

`osi_read_ready,` - The list of OSI associations and their current status (whether any data is waiting).

`osi_set_read_ready;` - The initial list with all entries set to their initial values.

`int osi_nfds=0;` – The number of OSI file descriptors placed in the `osi_read_ready` list. `PSelectMask()` function needs to know.

`DbvProtocolStack eProtocolStack;` - Stores the stack type for the association currently being accessed.

`DBVAssociateDetails *psAssociateDetails;` - Used to hold internal information about the current association. The physical file descriptors of the associations are read from this structure.

`struct PSAPindication *psPresentationIndication=NULL;` – Filled by ISODE when any errors occur.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              alAssocId[10];
DbvEventLocation       *psEventLocation = NULL;
DbvError               *psError = NULL;
DbvFds                  aulFds[10];

int place = 0;
              :
       /* Assume the alAssocId and aulFds arrays contain associations and file descriptors */
              :
       if(SIGetEventLocation((DbvAssocId **)&alAssocId, 1, (DbvFds **)&aulFds,
              &psEventLocation, &psError) == SI_OK)
        {
              if(psEventLocation != NULL)
              {
                     /* Receive a Z39.50 APDU over the communication link from the target */
                     place = 0;
                     while( psEventLocation->pplProviderEvent[place] != NULL)
                     {
                            printf("Got something from Association %d\n",*psEventLocation-
                                   >pplProviderEvent[place]);
                                   :
                            /* Do DbvReceiveDataOrigin here */
                                   :
                            place++;
                     }
                     place = 0;
                     while( psEventLocation->ppulUserEvent[place] != NULL)
                     {
                            /* Data from the host? */
                            /* service-user code here */
                     }
              }
              else
              {
                     printf("Timed out\n");
```

```
        }
    }
    else
    {
        /* Error in SIGetEventLocation */
        PrivateErrorHandling(psError);
        FreeSIErrorCode(&psError);
    }
```

9.2 SISetACSEFile() Function

Identification:

`int` **`SISetACSEFile`**`( char *pcACSEFile, DbvError **ppsError )`

Type:

This function is in file **si_utilities.c** and is contained in the library **libdbv**.

Purpose:

Changes the default "ACSE_configuration_File" to that of the one requested.

Function:

The function attempts to open the file for reading. If the open fails, the error **DBV_NAME_FILE** is returned in the `ppsError` structure and the return value is *SI_NOTOK*. Otherwise the filename is accepted and the global variable `DBVACSEFileName` is updated.

Subordinates:

None.

Dependencies:

None.

Interfaces:

This function effects the following global variables:

- `DBVACSEFileName` - This variable stores the filename of the ACSE Configuration File.

Resources:

None.

References:

Additional information on this function can be found in:

Section 14.0 concerning the ACSE configuration file.

D-008 API-Definition, 3.3.2

Processing:

Data:

`int return_value=SI_OK` – Stores the current return value of the function.

`FILE *psFd` – Holds the file descriptor of the file which being opened.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvError              *psError = NULL;

               :
               :

if(SISetACSEFile("My_ACSE_File", &psError) == SI_NOTOK)
{
       if(psError->eErrorClass == SI_Interface_Error &&
              psError->uErrorCode.lIfOsCoError==DBV_FILE)
       {
                     printf("The file does not exist\n");
       }
       else
       {
              PrivateErrorHandling(psError);
              FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
       }
}
else
       printf("The filename has been changed\n");

              :
              :
              :
```

9.3 SIWhichStack() Function

Identification:

```
int SIWhichStack( DbvAssocId          lAssocId,
                  DbvProtocolStack   *peProtocolStack,
                  DbvSIError        **ppsSIError )
```

Type:

This function is in file **si_utilities.c** and is contained in library **libdbv**.

Purpose:

Get the type of stack in use for the specified association.

Functionality:

The global list of associations is first checked to see if it is *NULL*. If it is not then the list is searched for the association specified. Once found, the stack used is placed in variable `peProtocolStack`. If there are no associations, the association cannot be found or the stack type is invalid, the error structure is completed and *SI_NOTOK* returned.

If successful, the function returns *SI_OK*.

Subordinates:

`DBVfill_error_structure()`

`SL_Locate()`

Dependencies:

The dependencies of this function are:

- The Association Id should be valid

Interfaces:

The function queries directly the global variable `DBVAssociateList`.

Resources:

None

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.3.3

Global variable information - section 11 of this document.

Processing:

None.

Data:

`DBVAssociateDetails *psAssociateDetails` – Used to store the result of a search of the linked list `DBVAssociateList`.

`int return_value = SI_OK` – Used to store the current return value of the function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId;
DbvProtocolStack        eProtocolStack;
DbvError               *psError = NULL;

               :
               :

        if(SIWhichStack(lAssocId, &eProtocolStack, &psError) == SI_OK)
        {
                if(eProtocolStack == TCP_Stack)
                {
                        printf("Association %d uses TCP/IP\n", lAssocId);
                }
                else if(eProtocolStack == OSI_Stack)
                {
                        printf("Association %d uses OSI\n", lAssocId);
                }
                else
                {
                        /* This will never happen */
                        printf("SIWhichStack has produced an error! (Not likely)\n");
                }
        }
        else
        {
                /* DBV_INVALIDSTYPE or DBV_INVALIDASSOC */
                PrivateErrorHandling(psError);
                FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
        }
```

9.4 SISetTraceParameter() Function

Identification:

```
int SISetTraceParameter(      DbvAssocId          lAssocId,
                              FILE               *psTraceFp,
                              DbvTraceLevel       eTraceLevel,
                              DbvSIError        **ppsSIError )
```

Type:

This function is in file **si_utilities.c** and is contained in library **libdbv**.

Purpose:

Specify the destination for trace information and the trace depth for all API functions.

Functionality:

Firstly, the global variable `DBVTraceList` is checked to see if it has been initialised. If not, then it is made to point to a newly constructed list.

Next see if the association Id given already has an associated trace file. If it does, then assign the new one pointed to by `psTraceFp`. Also set the new trace level.

If it doesn't then we are starting a new trace so allocate some memory for the trace details.

Now the association is checked to make sure that it is valid. Depending on the value of the input parameters the trace file is either added to the `DBVTraceList` linked list together with `eTraceLevel` and the association Id, the value of `eTraceLevel` changed for the given association, the trace file file descriptor changed or the trace details for the specified association removed from the list.

There are three trace levels:

- **Trace_Off -** No details written to trace file.
- **Trace_Low -** Log entry and exit to/from DBV-OSI II API functions, all errors and warnings, reception/sending of PDUs.
- **Trace_High -** As above, with the following additions. Function parameters (currently excluding full break down of APDU structures), and all state changes.

Note that it is the task of the service-user to open and close files.

Subordinates:

`SL_Create()`

`SIWhichStack()`

`DBVfill_error_structure()`

Dependencies:

The dependencies of this function are:

- The association Id should be valid.

- The variable `*psTraceFp` must point to a file opened for writing.

Interfaces:

The function manipulates the global linked list variable `DBVTraceList`.

Resources:

None

References:

Additional information on this function can be found in:

D-008 API-Definition, 3.3.4

Global variable information - section 11.0 of this document.

Processing:

None

Data:

`int return_value = SI_OK` – Holds the current return value of the function.

`int new_trace = FALSE` - This variable is used to store whether the association already has a

`ProtocolStack eProtocolStack` – Used when calling the `SIWhichStack()` function. Used in the function to write out the stack type to the trace file.

`DBVTraceDetails *psTraceInformation` – Structure used to hold the trace information for the specified association. Holds the file pointer, trace level and association Id. These structures are inserted into the linked list.

`time_t the_time` – The return value of the Unix system function `time()` which gives the current time is placed here.

`struct tm *time_struct` – The return value of the Unix system function `localtime()` is placed in here. This variable is the format taken by the `strftime()` function which is used to write the current time to a newly initialised trace file.

Usage:

```
#include <stdio.h>
#include <sys/types.h>

#include "sidefs.h"
#include "sitypes.h"

DbvAssocId              lAssocId;
DbvError               *psError = NULL;
DbvTraceLevel           eTraceLevel = Trace_High;
FILE                   *psTraceFp = NULL;

                :
                :
        sprintf(buffer, "target.%d.trace", getpid() );
        if((psTraceFp=fopen(buffer, "w")) == NULL)
        {
                printf("Couldn't open trace file!\n");
```

```
}
else
{
        if(SISetTraceParameter(lAssocId, psTraceFp,
                eTraceLevel, &psError)==SI_NOTOK)
        {
                PrivateErrorHandling(psError);
                FreeErrorCode(&psError);      /* release buffer, assign NULL pointer */
        }
}
        :
        :
```

9.5 Structure and Type Definitions for Utilitiy Functions

9.5.1 DbvAssocId:

```
typedef long      DbvAssocId
```

Used to reference a specific association.

9.5.2 DbvSecs:

```
typedef long      DbvSecs
```

Used to specify the length of time which the functions should block.

- CALL_BLOCKING to wait indefinitely
- CALL_ASYNCHRONOUS for return immediately
- <integer value> to wait for <integer value> seconds

9.5.3 DbvFds:

```
typedef unsigned long DbvFds
```

Used to declare file decriptors.

9.5.4 DbvEventLocation:

```
typedef struct DbvEventLocation {
      DbvFds      **ppulUserEvent;
      DbvAssocId  **pplProviderEvent;
} DbvEventLocation;
```

A structure of this type is returned from the `SIGetEventLocation()` function. It will contain all the associations and service-user file descriptors on which incoming data is waiting.

9.5.5 DbvProtocolStack:

```
typedef enum DbvProtocolStack {
            Unknown_Stack = 0,
            OSI_Stack = 1,
            TCP_Stack = 2
} DbvProtocolStack;
```

Returned from the SIWhichStack() function, it indicates the communication stack being used for the specified association.

9.5.6 DbvTraceLevel:

```
typedef enum DbvTraceLevel {
      Trace_Off = 0,
      Trace_Low = 1,
      Trace_High = 2
} DbvTraceLevel;
```

Used in function `SISetTraceParameter()` to set the trace level desired by the service-user for the association.

10 INTERNAL FUNCTIONS

The convention for functions which are internal to the API is that they will be prefixed with the letters DBV.

10.1 TCP/IP Association Control

10.1.1 DBVtcp_connect_to_host() Function

Identification:

```
int DBVTcpConnectToHost(      char                    *host,
                              ushort                   port,
                              DbvFds                  *socketfd,
                              DbvSecs                  lSeconds,
                              DbvAssociateState       *peAssociateState,
                              DBVAssociateDetails     *psAssociateDetails,
                              DbvError               **ppsError )
```

Type:

This function is in file **originassoc.c** which is contained in the library **libdbvorigin**.

Purpose:

The purpose of this function is to connect to the target using Unix TCP/IP connectivity functions.

Functionality:

The address of the target machine is specified in the variables host and port. If the TCP address was supplied in name format then the name is resolved using the `gethostbyname()` function. This function will check that the target machine exists. If the address was already in dot-number notation the validity of the address cannot be checked. A connection is attempted using the Unix system call `connect()`. The connection will be attempted for as long as specified in `lSeconds`. If the connection is unsuccessful the socket is deleted, otherwise the socket is returned in the `socketfd` variable.

Subordinates:

The subordinates to this function are:

`DBVfill_error_structure()`

Relevant Unix specific Communication functions:

`inet_addr()`

`htons()`

`gethostbyname()`

`socket()`

`connect()`

`close()`

Dependencies:

The dependencies of this function are:

- The machine must be connected to a TCP/IP network.

Interfaces:

The function will pass back to function DbvAssociateRequest() either SI_NOTOK and a completed errorstructure, or the result of the TCP `connect()`.

Resources:

This function will require the following resources:

- The Unix system TCP/IP libraries.

References:

Additional information on this function can be found in:

The Network programming guide for your flavour of Unix.

Processing:

None.

Data:

`struct sockaddr_in sina;` – Structure which hold the information required to connect to a host machine.

`struct hostent *host` – This is the return type of the `gethostbyname()` Unix function. It holds the details of the host (if it exists).

`int tempsocketfd` – Used to hold the socket file descriptor. Only if the function is successful is the file descriptor copied over to the socketfd variable.

`int return_value = SI_OK` – Used to store the current return value of the function.

10.2 OSI Association Control

10.2.1 DBVOsiConnectToHost() Function

Identification:

```
int DBVOsiConnectToHost(      DbvSecs                   lSeconds,
                              DbvTargetInformation     *psTargetInformation,
                              DbvAssociateState        *peAssociateState,
                              DBVAssociateDetails      *psAssociateDetails,
                              DbvAssociateResult      **ppuAssociateResult,
                              DbvError                **ppsError )
```

Type:

This function is in file **originassoc.c** which is contained in the library **libdbvorigin**.

Called from `DbvAssociateRequest()` function.

Purpose:

Initiate a connection to the specified target using ACSE services. Submit an A-ASSOCIATE.REQUEST action.

Functionality:

The function first attempts to obtain the service-user specified application context name and presentation contexts by calling the function `DBVGetACSEDetails()` (see section 10.3). If it returns *SI_NOTOK*, then the function does not attempt to complete the assocation and returns immediately.

If the `psTargetInformation` structure specifies a completed OSI address, then this is converted into a format which is compatible with ISODE. If the structure still specifies a name, then it is assumed that this is an entry in the **isoentities** file (the name will have been checked against entries in the **isoentities** file by the specify target by name functions - Section 10.4).

The ISODE function `str2aeinfo()` is called to create the application entity information, using the `pcTargetName` and `pcContext` variables. `aei2addr()` is then used to create the presentation address and `addr2ref()` to create the session reference. The quality of service is setup before calling either `AcAssocRequest()` for a blocking connection or `AcAsynAssocRequest()` for an asynchronous (or wait `lSeconds`) connection.

If the connection is not accepted or returns *NOTOK* (ISODE return value), then the function `DBVLogISODEACSEError()` is called to log the error to the origin system log.

If the function returned *OK*, *CONNECTING_1* or *CONNECTING_2* (ISODE return values), then the association Id assigned by ISODE is added to the `psAssociateDetails` structure.

If the function returned *CONNECTING_1* or *CONNECTING_2*, then `peAssociateState` is set to *Associate_Pending*.

If the association request returned *DONE* and was called asynchronous or returned *OK* and the call was blocking, then the internal function `DBVOsiCompleteDbvAssociateResult()` is called to complete the `ppuAssociateResult` structure.

Subordinates:

`DBVAllocCheck()`

`DBVAlloc()`

`DBVGetACSEDetails()`

`str2aeinfo()` – ISODE function

`aei2addr()` – ISODE function

`addr2ref()` – ISODE function

`AcAssocRequest()` – ISODE function

`AcAsynAssocRequest()` – ISODE function

`DBVLogISODEACSEError()`

`DBVWriteOriginLog()`

`DBVfill_error_structure()`

`DBVOsiCompleteDbvAssociateResult()`

Dependencies:

Function called from `DbvAssociateRequest()`.

Interfaces:

The function will pass back to function `DbvAssociateRequest()` either *SI_NOTOK* and a completed error structure, or the results of the OSI associate request.

Resources:

The function uses the following resources:

- Unix ISODE libraries.

References:

Additional information on this function can be found in:

ISODE Volume 10 Programmer's Guide: ASEs - Chapter 1.2 Associations

ISODE Volume 13 Programmer's Guide: Layer Interfaces - Chapter 1.2.2 (Presentation Services) Server Initialization, Chapter 2.3.2 (Session Services) Server Initialization.

Processing:

None.

Data:

`int return_value = SI_OK;` – Stores the current return value of the function.

`int lRequirements = DBV_SESSION_REQUIREMENTS;` – The session requirements. See `sil_defs.h` for the definition of `DBV_SESSION_REQUIREMENTS`.

`int lCount=0;` – Used to store the number of characters in the service-user provided OSI address (if completed) so that the correct amount of memory can be allocated for the address as one `char *`.

`char *pcTargetName = NULL;` – The service-user provided OSI address is copied here in a format correct for use in ISODE library functions.

`AEI psAppEntityInfo;` – Used within ISODE functions to store the application entity information.

`OID psContextOID=NULL;` – A structure to hold the application context object identifier in a format suitable for use in ISODE functions.

`char *pcContext=NULL;` – Holds the service-user specified application context (retrieved from the ACSE configuration files) before it is converted into an ISODE OID structure.

`struct AcSAPconnect *psAssocControlConnect = NULL;` – If the call to `Ac(Asyn)AssocRequest()` was successful, then this structure will contain information about the new association

`struct AcSAPindication *psAssocControlIndication = NULL;` – If the call to `Ac(Asyn)AssocRequest()` was unsuccessful or the association was rejected by the target, this structure will be filled out with diagnostic information.

`struct AcSAPabort *psAssocControlAbort = NULL;` – Points to the `AcSAPabort` structure which is contained within the `AcSAPindication` structure.

`struct PSAPaddr *psPresentationAddress = NULL;` – Holds the presentation address of the target which is obtained from the application entity information (using function `aei2addr()`).

`struct PSAPctxlist *psPresentationContextList = NULL;` – The service-user requested presentation contexts (as supplied in the ACSE configuration file) are converted into ISODE OIDs and placed into this structure.

`struct SSAPref *psSessionRef;` – This is the connection identifier used by the origin. It created with a call to `addr2ref(PLocalHostName())` where `PLocalHostName()` is an ISODE function which returns in char * format the name of the local host machine.

`struct SSAPref sSessionRef;` – If the call to `addr2ref()` fails, then `psSessionRef` is made to point to this structure, the contents of which are then zeroed.

`struct QOStype sQOS;` – Used to hold the quality of service required.

10.2.2 DBVOsiCompleteDbvAssociateResult() Function

Identification:

```
int DBVOsiCompleteDbvAssociateResult(struct AcSAPconnect *psAssocControlConnect,
                                     struct AcSAPabort   *psAssocControlAbort,
                                     DbvAssociateResult **ppuAssociateResult,
                                     DbvAssociateState   *peAssociateState,
                                     DbvError          **ppsError)
```

Type:

This function is in file **originassoc.c** which is contained in the library **libdbvorigin**.

Purpose:

Completes the `ppuAssociationResult` and `peAssociateState` variables with information detailing the state of the association and the response from the target (if there is any).

10.3 ACSE Configuration File Functions

10.3.1 DBVGetACSEDetails() Function

Identification:

```
int DBVGetACSEDetails(  char                    **ppcContext,
                        OID                      *ppsContextOID,
                        struct PSAPctxlist      **ppsPresentationContextList,
                        DbvError                **ppsError)
```

Type:

This function is in file **originassoc.c** which is contained in the library **libdbvorigin**.

Purpose:

Retrieves the desired application context and presentation context list from the ACSE configuration file.

Functionality:

A call to function `DBVGetACN()` retrieves the application context name in `char *` format and `DBVGetPCDL()` returns a completed `psPresentationContextList` structure. The application context name is then converted into an ISODE specific OID.

Subordinates:

`DBVAllocCheck()`

`DBVAlloc()`

`DBVGetACN()`

`DBVGetPCDL()`

`str2oid() - ISODE function`

`oid_cpy() - ISODE function`

`DBVfill_error_structure()`

Dependencies:

Called from function `DBVOsiConnectToHost()`.

10.3.2 DBVGetACN() Function

Identification:

```
int DBVGetACN(    char        **ppcContext,
                  DbvError    **ppsError )
```

Type:

This function is in file **ACSE.c** which is contained in the library **libdbv**.

Purpose:

This function retrieves the Application Context Name from the ACSE configuration file.

Functionality:

The function first checks that the configuration file exists, if so it reads in records until it finds the header "`Application_Context_Name`". The function `DBVReadACNs()` is called which returns the context which is to be used as the application context name, or *SI_NOTOK* and a completed error structure if an error occurred.

If the entry "`Application_Context_Name`" is not found in the file then the default context "`1.0.10163.1.1`" (SR service oid) is used.

If the function is successful, it places the context name in the `ppcContext` variable.

Subordinates:

`TBL_Create()`

`DBVWriteLog()`

`TBL_GetRecord()`

`DBVfill_error_structure()`

`DBVReadACNs()`

`DBVAllocCheck()`

`TBL_Destruct()`

Interfaces:

Called from function `DBVGetACSEDetails()` (Section 10.3.2)

10.3.2 DBVReadACNS(), DBVConvertToOID() and DBVValidateACN() Functions

Identification:

```
int DBVReadACNs(  TBL           InTbl,
                  char        **ppcContext,
                  DbvError    **ppsError )


char *DBVConvertToOID( char *pcTempContext )

int DBVValidateACN(     char        **ppcACNs,
                        DbvError    **ppsError )
```

Type:

This function is in file **ACSE.c** which is contained in the library **libdbv**.

Purpose:

Returns the context to be used in the association in `char*` oid format (either `1.0.10163.1.1` or `1.2.840.10003.1.1`)

Functionality:

Reads in the context name(s) specified in their word format (either `"Basic_Z3950_Application_Context"` or `"Basic_SR_Application_Context"`). They are then converted into string format oids by the function `DBVConvertToOID()` (e.g. `"1.0.10163.1.1"`). Any which cannot be converted are left as they are.

The list of `char *` contexts is now validated by the function `DBVValidateACN()`, which returns one string, which is the context to be used (or *SI_NOTOK* and an error if one occurred).

The function `DBVValidateACN()` sets the global variables DBVbZ3950 and DBVbSR to DBV_TRUE if `"Basic_Z3950_Application_Context"` or `"Basic_SR_Application_Context"` are in the list. This is so that the oids can be added to the presentation context definition list automatically if they were present in the application context list.

10.3.2 DBVGetPCDL(), DBVReadPCDLs() and DBVAddTheApplicationContexts() Functions

Identification:

```
int DBVGetPCDL(   struct PSAPctxlist       *psPresentationContextList,
                  DbvError                **ppsError )

int DBVReadPCDLs( TBL                       InTbl,
                  struct PSAPctxlist       *psPresentationContextList,
                  DbvError                **ppsError)

int DBVAddTheApplicationContexts(struct PSAPctxlist *psPresentationContextList,
                                 DbvError          **ppsSIError)
```

Type:

This function is in file **ACSE.c** which is contained in the library **libdbv**.

Purpose:

This set of functions read in from the ACSE configuration file the desired presentation contexts, add the specified application context(s) and return them in a completed `psPresentationContextList`.

10.4 Specifying Target by Name Functions

10.4.1 DBVConvertNameToAddress() and DBVCompleteAddress()

Identification:

```
int DBVConvertNameToAddress(  char                      *pcTargetName,
                              DbvTargetInformation     **ppsTargetInformation,
                              DbvError                 **ppsError )

int DBVCompleteAddress( DbvTargetInformation    **ppsTargetInformation,
                        TBL                       InTbl,
                        TBL_Return                T_Return,
                        DbvError                **ppsError)
```

Type:

This function is in file **address_by_name.c** which is contained in the library **libdbvorigin**.

Purpose:

This group of functions attempt to resolve the target name specified into a completed address.

Functionality:

The file target_names.tbl contains mappings between names and full TCP addresses. If the specified entry exists in the file, then the `ppsTargetInformation` structure is completed with the full target address.

If the name given is not in the list, then the ISODE services database is searched for a corresponding entry. If the entry is found in the database, then the function returns *SI_OK* and `*ppsTargetInformation` pointing to *NULL*. This is because the ISODE functions in `DBVOsiCOnnectToHost()` will search the database automatically before the connect stage.

If the specified name does not appear either file, an error is returned (*DBV_NAME_ENTRY*) and the function returns *SI_NOTOK*.

10.5 State Machine Functions

10.5.1 DBVChangeState() Function

Identification:

```
int DBVChangeState(     DbvAssocId lAssocId,
                        long       lEvent,
                        DbvError **ppsError )
```

Type:

This function is in file **state_machine.c** which is contained in the library **libdbv**.

Purpose:

Attempts to change the state of the specified association using the event supplied.

Functionality:

The association is checked to see whether it belongs to a target or origin application. Depending on which, either the target or origin state machine table is used and `lEvent` refers to either an incoming origin or target event.

The association's current state is read from the `psAssociateDetails` structure and the event applied to it. In the event of an error, either the `DBVOriginSMError()` or `DBVTargetSMError()` functions are called.

Subordinates:

`SIWhichStack()`

`DBVOriginSMError()`

`DBVTargetSMError()`

10.5.2 DBVOriginSMError(), DBVTargetSMError() Functions

Identification:

```
int DBVOriginSMError(   DbvAssocId          lAssocId,
                        long                lEvent,
                        DbvError          **ppsError)

int DBVTargetSMError(   DbvAssocId          lAssocId,
                        long                lEvent,
                        DbvError          **ppsError)
```

Type:

These functions are in file **state_machine.c** which is contained in the library **libdbv**.

Purpose:

Produces a completed error structure detailing the error which occurred in the state machine.

Functionality:

Formulates a string which explains to the service user which event occurred on which state and that it was invalid.

Subordinates:

None.

Dependencies:

None.

10.6 Sending/Receiving Data

10.6.1 DBVencode_and_send() Function

Identification:

```
int DBVencode_and_send( PDU                *PDU_ptr,
                        DbvAssocId          lAssocId,
                        DbvError          **ppsError )
```

Type:

This function is in file **z3950_send_receive.c** which is contained in the library **libdbv**.

Purpose:

Encodes the snacc PDU structure into a binary string and sends it to the target.

Functionality:

The Provided PDU is encoded into a binary string by the function `DBVencode_pdu_to_bytes()`. If this is successful, then the stack type of the association is checked.

For OSI associations the function turns the data into a presentation element and sends with a call to the isode function `PDataRequest()`. For TCP, the function `DBVsend_data_over_socket()` is called. This performs the Unix TCP command `send()` until the number of bytes sent equals the number of bytes in the encoded PDU.

Subordinates:

`DBVEnterFuncTrace()`

`DBVAllocCheck()`

`DBV_reset_external_counters()`

`DBVTraceLevel()`

`DBVTrace()`

`DBVencode_pdu_to_bytes()`

`SIWhichStack()`

`SL_Current()`

`DBVsend_data_over_socket()`

`oct2prim()` – ISODE function

`AcFindPCI()` – ISODE function

`PDataRequest()` – ISODE function

`DBVLeaveFuncTrace()`

Dependencies:

Function called from each pdu request and response function.

10.6.3 DBVreceive_data_over_socket() Function

Identification:

```
int DBVreceive_data_over_socket(    DbvAssocId          lAssocId,
                                    PDU                *PDU_ptr,
                                    int                 socket,
                                    DbvSecs             lSeconds,
                                    char               **ber_data,
                                    unsigned long       *ber_data_length,
                                    DbvError           **ppsError)
```

Type:

This function is in file **z3950_send_receive.c** which is contained in the library **libdbvorigin**.

Purpose:

Receive data from the association peer. Once data is being received, wait for a complete PDU.

Functionality:

The function first sets the value of the `TimeOut` structure depending on the value of `lSeconds` (set by the service-user) and places the socket file descriptor in a mask variable suitable for the Unix cammand `select()`. The function then performs a `select()` on the socket file descriptor associated with the association. What happens next depends on the return value of the `select()` call.

If `select()` returned `-1`, then an error occurred which is dealt with accordingly. If it returned `0`, then `select()` timed out and no data was received. If this is the case, then the function returns the value 5 (indicating *SI_OK* and socket timed out with nothing to read) to the calling function.

If data has been detected, then the `recv()` function is called to receive the incoming PDU. The data is received in packets which have a maximum size of 4096 bytes. This is the maximum size on most Unix systems. After each read, the data currently read is checked to see if it is a complete PDU. This is done by a call to the function `IsCompleteBER()`. If a complete PDU has been received, then it is decoded by calling the function `DBVdecode_bytes_to_pdu()`. The decoded PDU is then passed back to the calling function.

Subordinates:

`DBVfill_error_structure()`

`IsCompleteBER() - See file berutil.c`

`DBVdecode_bytes_to_pdu()`

Dependencies:

Called by the functions `DbvReceiveDataOrigin()` and `DbvReceiveDataTarget()`.

10.6.4 DBVreceive_data_over_OSI() Function

Identification:

```
int DBVreceive_data_over_OSI( DbvAssocId          lAssocId,
                              PDU                *PDU_ptr,
                              int                 lOSIAssocId,
                              DbvSecs             lSeconds,
                              char               **ber_data,
                              unsigned long       *ber_data_length,
                              DbvError           **ppsError )
```

Type:

This function is in file **z3950_send_receive.c** which is contained in the library **libdbv**.

Purpose:

Receive data from the association peer. Once data is being received, wait for a complete PDU.

Functionality:

This function follows the same basic structure as `DBVreceive_data_over_tcp()`.

The function first sets the value of the `TimeOut` structure depending on the value of `lSeconds` (set by the service-user) and places the OSI file descriptor in a mask variable suitable for the Unix cammand `xselect()` (ISODE equivalent to select()). This is done using the ISODE function `PSelectMask()`. The function then performs an `xselect()`. What happens next depends on the return value of the `xselect()` call.

If xselect() returned NOT_OK, then an error occurred which is dealt with accordingly. If it returned OK, then xselect() timed out and no data was received. If this is the case, then the function returns the value 5 (indicating *SI_OK* and socket timed out with nothing to read) to the calling function.

If data has been detected, then the PReadRequest() function is called to receive the incoming PDU. After each read, the data currently read is checked to see if it is a complete PDU. This is done by a call to the function IsCompleteBER(). If a complete PDU has been received, then it is decoded by calling the function DBVdecode_bytes_to_pdu(). The decoded PDU is then passed back to the calling function.

Subordinates:

`DBVEnterFuncTrace()`

`DBVAllocCheck()`

`DBVAlloc()`

`PSelectMask()`

`xselect()`

`DBVfill_error_structure()`

`PReadRequest()`

`IsCompleteBER()`

`DBVdecode_bytes_to_pdu()`

`DBVLeaveFuncTrace()`

Dependencies:

Called from API functions `DbvReceiveDataTarget()` and `DbvReceiveDataOrigin()`.

10.7 Trace File Control functions

10.7.1 DBVWrite() Function

Identification:

```
void DBVWrite(    char   *data,
                  FILE   *fd )
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

Writes a string to a file and flushes the buffer. Used so that information is written to the file instantaneously.

Functionality:

Uses `puts()` to place the data in the file buffer then `fflush()`es it to the file.

10.7.2 DBVTraceLevel() Function

Identification:

```
TraceLevel DBVTraceLevel(     AssocId lAssocId );
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

Returns the trace level for the given association Id. This function is used in many of the API components to decide whether trace information needs to be written.

Functionality:

If there are no trace details in the `DBVTraceList` linked list or the association has no corresponding trace details then *`Trace_Off`* is returned. Otherwise the value of `psTraceInformation->DBVTraceLevel` (the trace level for the given association Id) is returned.

Subordinates:

The subordinates to this function are:

SL_Locate()

Interfaces:

This function uses the global variable `DBVTraceList`.

References:

Additional information on this function can be found in:

Global variable information - Section 11.0 of this document.

Data:

`DBVTraceDetails *psTraceInformation` – This structure is used to hold the trace information extracted from the linked list `DBVTraceList`.

10.7.3 DBVTrace() Function

Identification:

```
int DBVTrace(     AssocId       lAssocId,
                  int           level,
                  char         *trace )
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

Writes the trace information contained in `trace` to a trace file. Used by many components of the API. Contains logic to only write data when certain trace levels are specified.

Functionality:

Firstly, the global linked list `DBVTraceList` is checked for *`NULL`*. If any trace details can be found for the specified association then the details are written to the trace file, unless the trace level is *`Trace_Off`* or `level` is *`Trace_High`* and `psTraceInformation->DBVTraceLevel` is *`Trace_Low`*.

This function always returns *`TRUE`*.

Subordinates:

`SL_Locate()`

Interfaces:

This function uses the global variable `DBVTraceList`.

References

Additional information on this function can be found in:

Global variable information - Section 11.0 of this document.

Data:

`DBVTraceDetails *psTraceInformation` – This structure is used to hold the trace information extracted from the linked list `DBVTraceList`.

10.7.4 DBVErrorTrace() Function

Identification:

```
void DBVErrorTrace(     AssocId       lAssocId,
                        char         *pcDiagnostic,
                        SIError     **ppsSIError )
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

Writes an error structure to a trace file. Called by `DBVfill_error_structure()` (Section X.X.X). If possible, write out the error which occurred to the trace file.

Functionality:

If the association Id specified has an associated trace file, then write the error information to the trace file.

Subordinates:

The subordinates to this function are:

`DBVTraceLevel()`

`DBVTrace()`

References:

Additional information on this function can be found in:

Global variable information - Section 11.0 of this document.

Data:

`SIError *psSIError` – Used to de-reference `ppsSIError` one level to improve code readability.

10.7.5 DBVEnterFuncTrace() function

Identification:

```
void DBVEnterFuncTrace( char          *function_name,
                        AssocId        lAssocId )
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

If there is a trace file associated with the association this function writes a message to it indicating that a function has been entered. Used by many API components.

Functionality:

Check if the provided association Id has an open trace file. If so, append the name of the function to the message "Entering function: " and write it to the trace file.

Subordinates:

`DBVTraceLevel()`

`DBVTrace()`

10.7.6 DBVLeaveFuncTrace() Function

Identification:

```
void DBVLeaveFuncTrace( char          *function_name,
                        AssocId        lAssocId,
                        int            return_value )
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

If there is a trace file associated with the association this function writes a message to it indicating that the function is about to be left. Used at the end of many API components.

Functionality:

Check if the provided association Id has an open trace file. If so, append the name of the function to the message "Leaving function: " and write it to the trace file. Then write either "SI_OK" or "SI_NOTOK" to the trace file prepended with "Function returned: ".

10.8 Service-Provider System Log File Functions

10.8.1 DBVWriteOriginLog(), DBVWriteTargetLog(), DBVWriteLog(), DBVWriteErrorToLog(), DBVLogISODEACSEError() and DBVLogISODEPError() Functions

Identification:

```
void DBVWriteOriginLog( char *pcLogData)

void DBVWriteTargetLog( char *pcLogData)

void DBVWriteLog(       char *pcLogData)

void DBVWriteErrorToLog( DbvError   **ppsError,
                         char        *pcDiagnostic)

void DBVLogISODEACSEError(    char                    *pcFunction,
                              struct AcSAPabort       *psAssocControlAbort)

void DBVLogISODEPError(       char                    *pcFunction,
                              struct PSAPabort        *psPresentationAbort)
```

Type:

This function is in file **logfiles.c** which is contained in the library **libdbv**.

Purpose:

Write information out to the system API log files, where writing to trace files is impossible or inappropriate.

Functionality:

When a log file function is entered the log file is tested to see if it is locked. If not, the messages supplied are written to the file. If the file is locked then the function waits for a short time before retrying. A function will retry five times before giving up. If a function gives up, the messages are written to `stderror`.

The default logfile filename to be used is specified in the `sil_defs.h` header file.

10.9 Miscellaneous Functions.

10.9.1 DBVfill_out_event_location_structure() Function

Identification:

```
void DBVfill_out_event_location_structure( DbvAssocId **pplAssocId,
                                           DbvFds **ppulFileDescriptor,
                                           fd_set *pread_ready,
                                           fd_set *posi_read_ready,
                                           DbvEventLocation **ppsEventLocation );
```

Type:

This function is in file **si_utilities.c** which is contained in the library **libdbv**.

Purpose:

Called from API function `SIGetEventLocation()`.

Fills out the given `ppsEventLocation` structure with associations and file descriptors which are ready for reading.

Functionality:

Firstly the function counts the number of Association Id's which are ready for reading (by checking if the association's file descriptor is set in the `pread_ready/posi_read_ready` lists of descriptors. This is done using the `FD_SET()` unix macro) and allocates enough memory (using the `DbvAlloc()` macro defined in **dbv_proto.h**) for the `pplProviderEvent` pointers of the `ppsEventLocation` structure. Next the service-user file descriptors are checked (again using `pread_ready` and `FD_SET()`) and enough memory for the `ppulUserEvent` pointers of the `ppsEventLocation` structure is allocated.

The association Id's of the selected associations are then placed in the `pplProviderEvent` area of the `ppsEventlLocation` structure. Next, the selected file descriptors are placed in the `ppulUserEvent` list of the `ppsEventLocation` structure. Memory is allocated as each variable is selected for inclusion.

Each list is *NULL* terminated.

The structure now indicates which associations and service-user file descriptors have incoming events pending.

Subordinates:

The subordinates to this function are:

SL_Locate()

DbvAlloc() - Macro defined in **dbv_proto.h**

Dependencies:

Function called from `SIGetEventLocation()`

Interfaces:

Uses the global linked list `DBVAssociateList` to retrieve assocation details.

Resources:

This function will use the following resources:

- TCP/IP Unix libraries
- OSI (ISODE) Unix libraries

References:

Additional information on this function can be found in:

The Network programming guide for your flavour of Unix.

The ISO Development Environment: User's Manual. Volume 1: Application Services & Volume 2: Underlying Services.

Processing:

None.

Data:

`int count = 0` – Used as a subscript for indexing arrays.

`int Eventcount = 0` – Used to count the number of associations or descriptors which are ready for reading.

`EventLocation *psLocalEventLocation = *ppsEventLocation` – Used to dereference ppsEventLocation by one level for increased readiblity.

`DBVAssociateDetails *psAssociateDetails` – Stores the details of the association. Used to access the actual TCP or OSI connection.

10.10 PDU Checking And Mapping Subordinate Functions

SR-/Z39.50 Subordinate Functions

Overview:

The subordinates of each PDU function follow a similar design and structure. As shown in the sections above, there are three main top subordinate functions called by each API PDU function.

1) A function to check the fields in the DBV PDU structure passed as a parameter.

2) A function to map the DBV PDU fields onto similar fields in a 'Snacc' PDU structure.

3) A function to encode and send the 'Snacc' PDU structure.

Each of these top subordinate functions is the root of a tree of additional subordinate functions.

Dbv Structure Checking:

The first subordinate called by each PDU function is a function which, through a tree structure of additional subordinates, checks all DBV PDU fields as possible. This checking (parsing) is designed to detect any errors which would cause fatal errors during DBV->Snacc PDU structure mapping. There are seven categories of checking which are used at multiple points during the checking.

1) Compulsory field checking. All compulsory fields which are pointers are checked to ensure that they are not NULL.

2) Compulsory union checking. Compulsory unions are checked to ensure that a valid union member is spcified and that the specified union member specifed exists.

3) Compulsory list checking. Compulsory lists which are pointers are checked to ensure that they are not NULL.

4) List contents checking. If a list is compulsory or is optional and exists, the first item is checked to ensure that it is not NULL - a list cannot be empty, its existence implies one or more items.

5) Object Identifier checking. Object Identifier fields are checked to ensure that they contain a valid object identifier for their context.

6) Enumeration field checking. Enumeration type fields are checked to ensure that the contain a value in the range of the enumeration.

7) DbvBool field checking. DbvBool type fields are checked to ensure that they contain a valid DbvBool value.

The top level DBV PDU field functions called by each PDU function are listed below:

Origin Functions:

DbvInitializeRequest (in file origin_z3950_init.c) : DBVcheckDbvInitReqFields()

DbvSearchRequest (in file origin_z3950_search.c): DBVcheckDbvSrchReqFields()

DbvPresentRequest (in file origin_z3950_present.c) : DBVcheckDbvPresReqFields()

DbvDeleteResultSetRequest (in file origin_z3950_delete.c): DBVcheckDbvDelReqFields()

DbvResourceReportRequest (in file origin_z3950_resrep.c): DBVcheckDbvResRepReqFields()

DbvScanRequest (in file origin_z3950_scan.c): DBVcheckDbvScanReqFields()

DbvSortRequest (in file origin_z3950_sort.c): DBVcheckDbvSortReqFields()

DbvExtendedServicesRequest (in file origin_z3950_extend.c): DBVcheckDbvExtendReqFields()

Target functions:

DbvInitializeResponse (in file target_z3950_init.c): DBVcheckDbvInitRspFields()

DbvSearchResponse (in file target_z3950_search.c): DBVcheckDbvSrchRspFields()

DbvPresentResponse (in file target_z3950_present.c): DBVcheckDbvPresRspFields()

DbvDeleteResultSetResponse (in file target_z3950_delete.c): DBVcheckDbvDelRspFields()

DbvResourceReportResponse (in file target_z3950_resrep.c): DBVcheckDbvResRepRspFields()

DbvScanResponse (in file target_z3950_scan.c): DBVcheckDbvScanRspFields()

DbvSortResponse (in file target_z3950_sort.c): DBVcheckDbvSortRspFields()

DbvSegmentRequest (in file target_z3950_segment.c): DBVcheckDbvSegReqFields()

DbvExtendedServicesResponse (in file target_z3950_extend.c): DBVcheckDbvExtendRspFields()

Origin/Target Shared Functions:

DbvCloseRequest (in file shared_z3950_close.c): DBVClose(), DBVcheckDbvCloseReqFields()

DbvCloseResponse (in file shared_z3950_close.c): DBVClose(), DBVcheckDbvCloseReqFields()

All these top level functions contains calls to seperate checking functions for each field in the structure which needs checking.

Each of these additional checking functions may themselves have additional subordinate checking functions.

Except in trival cases, each structure within each field is checked within its own subordinate function.

All checks are made by simply accessing a relevant field and examining its content.

Function DBVCheckDbvInitReqFields() from file origin_z3950_init.c is reproduced below together with the first subordinate it calls - DBVcheck_dbv_initreq_prover() - which checks the protocol version field of the DBV InitializeRequest structure. The reference id field, the first field in the PDU, is not checked as the DBV->Snacc mapping functions cope with any DbvOctetString format.

```
int DBVCheckDbvInitReqFields(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
{

        if (DBVcheck_dbv_initreq_prover(lAssocId, psInitializeReq, ppsSIError) == SI_NOTOK)
                return SI_NOTOK;

        if (DBVcheck_dbv_initreq_options(lAssocId, psInitializeReq, ppsSIError) == SI_NOTOK)
                return SI_NOTOK;

        if (DBVcheck_dbv_initreq_auth(lAssocId, psInitializeReq, ppsSIError) == SI_NOTOK)
                return SI_NOTOK;

        if (DBVcheck_dbv_initreq_userinf(lAssocId, psInitializeReq, ppsSIError) == SI_NOTOK)
                return SI_NOTOK;


        if (DBVcheck_dbv_initreq_otherinfo(lAssocId, psInitializeReq, ppsSIError) == SI_NOTOK)

                return SI_NOTOK;

        return SI_OK;
}

int DBVcheck_dbv_initreq_prover(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
{

        /* If field doesnt exist */
        if(psInitializeReq->pcProtocolVersion == NULL)
                return(DBVdbv_field_missing_error(lAssocId, "psInitializeReq->pcProtocolVersion",
ppsSIError));


        /* pcProtocolVersion exists - check its value */
        if (DBVcheck_bit_string(psInitializeReq->pcProtocolVersion) == SI_NOTOK)
                return(DBVdbv_bit_string_error(lAssocId, "psInitializeReq->pcProtocolVersion",
psInitializeReq->pcProtocolVersion, ppsSIError));

        return SI_OK;
}
```

All field checks return either *SI_OK* or *SI_NOTOK*.

If all field checks pass, the top level checking function returns *SI_OK*. If any field error is detected at any point during the checking, the error structure passsed down as a parameter to the checking functions is completed with diagnostics and the function detecting the error returns *SI_NOTOK*. This return causes higher level checking functions to return with the same error comndition until eventually the top level checking function returns with *SI_NOTOK*. Note that during error structure filling, string diagnostics are formed using SIL developed functions sa_strcpy and sa_strcat() in z3950_shared.c - safe (robust) versions of strcpy() and strcat() which will never crash due to buffer size problems. Using these functions is necessary because of the possiblity of recursion within PDU fields.

Dbv->Snacc Structure Mapping:

After DBV PDU field checking a DBV->'Snacc' PDU field mapping function is called.

All DBV PDU structures are specified in header file sitypes.h.

Several files in DBV_OSI II, all those beginning with z39_50, were automatically generated from a specially prepared version of the Z39.50 Version 3 specification held only at SIL using an ASN.1-to-C generator utility called snacc1.1.

Snacc generated files contain C structures corressponding to the entire ASN.1 used as input together with 'BER' encoding and decoding routines for all these structures. BER encoded data is the low level format of Z39.50.

A fragment of the PDU structures in both sitypes.h and snacc generated file Z39_50_apdu_1994.h are shown below:

sitypes.h:

```
/* Structures associated with SR-/Z39.50 functions */

/* For the SR-/Z39.50 service request and response structures.refer to the
annex. */

typedef struct DbvOriginData {
  DbvOriginEvent              eOriginEvent;
  union {
    DbvInitializeRsp          *psInitializeRsp;
    DbvSearchRsp              *psSearchRsp;
    DbvPresentRsp             *psPresentRsp;
    DbvDeleteResultSetRsp     *psDeleteResultSetRsp;
    DbvScanRsp                *psScanRsp;
    DbvResourceReportRsp      *psResourceReportRsp;
    DbvSegmentReq             *psSegmentReq;
    DbvSortRsp                *psSortRsp;
    DbvCloseReq               *psCloseReq;
    DbvExtendedServicesRsp    *psExtendedServicesRsp;
/*     DbvReleaseReq            *peReleaseReq;   Target is not allowed to close the
                                                 association */
    DbvOctetString            *psInvalidAPDU;
  }     uOriginData;
} DbvOriginData;

typedef struct DbvTargetData {
  DbvTargetEvent       eTargetEvent;
  union {
    DbvInitializeReq          *psInitializeReq;
    DbvSearchReq              *psSearchReq;
    DbvPresentReq             *psPresentReq;
    DbvDeleteResultSetReq     *psDeleteResultSetReq;
    DbvScanReq                *psScanReq;
    DbvResourceReportReq      *psResourceReportReq;
    DbvSortReq                *psSortReq;
    DbvCloseReq               *psCloseReq;
    DbvExtendedServicesReq   *psExtendedServicesReq;
    DbvReleaseReq             *peReleaseReq;
    DbvOctetString            *psInvalidAPDU;
  }                    uTargetData;
} DbvTargetData;
```

z39_50_apdu_1994.h:

```
typedef struct PDU /* CHOICE */
{
    enum PDUChoiceId
    {
        PDU_INITREQUEST,
        PDU_INITRESPONSE,
        PDU_SEARCHREQUEST,
        PDU_SEARCHRESPONSE,
        PDU_PRESENTREQUEST,
        PDU_PRESENTRESPONSE,
        PDU_DELETERESULTSETREQUEST,
        PDU_DELETERESULTSETRESPONSE,
        PDU_ACCESSCONTROLREQUEST,
        PDU_ACCESSCONTROLRESPONSE,
```

```
        PDU_RESOURCECONTROLREQUEST,
        PDU_RESOURCECONTROLRESPONSE,
        PDU_TRIGGERRESOURCECONTROLREQUEST,
        PDU_RESOURCEREPORTREQUEST,
        PDU_RESOURCEREPORTRESPONSE,
        PDU_SCANREQUEST,
        PDU_SCANRESPONSE,
        PDU_SORTREQUEST,
        PDU_SORTRESPONSE,
        PDU_SEGMENTREQUEST,
        PDU_EXTENDEDSERVICESREQUEST,
        PDU_EXTENDEDSERVICESRESPONSE,
        PDU_CLOSE,
        PDU_UNKNOWN
    } choiceId;
    union PDUChoiceUnion
    {
    struct InitializeRequest* initRequest; /* [20] IMPLICIT InitializeRequest */
    struct InitializeResponse* initResponse; /* [21] IMPLICIT InitializeResponse */
    struct SearchRequest* searchRequest; /* [22] IMPLICIT SearchRequest */
    struct SearchResponse* searchResponse; /* [23] IMPLICIT SearchResponse */
    struct PresentRequest* presentRequest; /* [24] IMPLICIT PresentRequest */
    struct PresentResponse* presentResponse; /* [25] IMPLICIT PresentResponse */
    struct DeleteResultSetRequest* deleteResultSetRequest; /* [26] IMPLICIT DeleteResultSetRequest
*/
    struct DeleteResultSetResponse* deleteResultSetResponse; /* [27] IMPLICIT
DeleteResultSetResponse */
    struct AccessControlRequest* accessControlRequest; /* [28] IMPLICIT AccessControlRequest */
    struct AccessControlResponse* accessControlResponse; /* [29] IMPLICIT AccessControlResponse */
    struct ResourceControlRequest* resourceControlRequest; /* [30] IMPLICIT ResourceControlRequest
*/
    struct ResourceControlResponse* resourceControlResponse; /* [31] IMPLICIT
ResourceControlResponse */
    struct TriggerResourceControlRequest* triggerResourceControlRequest; /* [32] IMPLICIT
TriggerResourceControlRequest */
    struct ResourceReportRequest* resourceReportRequest; /* [33] IMPLICIT ResourceReportRequest */
    struct ResourceReportResponse* resourceReportResponse; /* [34] IMPLICIT ResourceReportResponse
*/
    struct ScanRequest* scanRequest; /* [35] IMPLICIT ScanRequest */
    struct ScanResponse* scanResponse; /* [36] IMPLICIT ScanResponse */
    struct SortRequest* sortRequest; /* [43] IMPLICIT SortRequest */
    struct SortResponse* sortResponse; /* [44] IMPLICIT SortResponse */
    struct Segment* segmentRequest; /* [45] IMPLICIT Segment */
    struct ExtendedServicesRequest* extendedServicesRequest; /* [46] IMPLICIT
ExtendedServicesRequest */
    struct ExtendedServicesResponse* extendedServicesResponse; /* [47] IMPLICIT
ExtendedServicesResponse */
    struct Close* close; /* [48] IMPLICIT Close */
    } a;
} PDU;

AsnLen BEncPDU PROTO((BUF_TYPE b, PDU* v));

void BDecPDU PROTO(( BUF_TYPE b, PDU* result, AsnLen* bytesDecoded, ENV_TYPE env));
AsnLen BEncPDUContent PROTO((BUF_TYPE b, PDU* v));

void BDecPDUContent PROTO(( BUF_TYPE b, AsnTag tagId0, AsnLen elmtLen0, PDU* v, AsnLen*
bytesDecoded, ENV_TYPE env));


void PrintPDU PROTO((FILE* f, PDU* v, unsigned short int indent));
void FreePDU PROTO((PDU* v));
```

The top level structure mapping functions called by each PDU function are listed below:

Origin Functions:

DbvInitializeRequest (in file origin_z3950_init.c) : DBVsetup_snacc_initrequest()

DbvSearchRequest (in file origin_z3950_search.c): DBVsetup_snacc_srchrequest()

DbvPresentRequest (in file origin_z3950_present.c) : DBVsetup_snacc_presrequest()

DbvDeleteResultSetRequest (in file origin_z3950_delete.c): DBVsetup_snacc_delrequest()

DbvScanRequest (in file origin_z3950_scan.c): DBVsetup_snacc_scanrequest()

DbvSortRequest (in file origin_z3950_sort.c): DBVsetup_snacc_sortrequest()

DbvExtendedServicesRequest (in file origin_z3950_extend.c): DBVsetup_snacc_extendrequest()

Target functions:

DbvInitializeResponse (in file target_z3950_init.c): DBVsetup_snacc_initresponse()

DbvSearchResponse (in file target_z3950_search.c): DBVsetup_snacc_searchresponse()

DbvPresentResponse (in file target_z3950_present.c): DBVsetup_snacc_presresponse()

DbvDeleteResultSetResponse (in file target_z3950_delete.c): DBVsetup_snacc_delresponse()

DbvScanResponse (in file target_z3950_scan.c): DBVsetup_snacc_scanresponse()

DbvSortResponse (in file target_z3950_sort.c): DBVsetup_snacc_sortresponse()

DbvSegmentRequest (in file target_z3950_segment.c): DBVsetup_snacc_segrequest()

DbvExtendedServicesResponse (in file target_z3950_extend.c):
DBVsetup_snacc_extendresponse()

Origin/Target Shared Functions:

DbvCloseRequest (in file shared_z3950_close.c): DBVClose(), DBVsetup_snacc_closerequest()
DbvCloseResponse (in file shared_z3950_close.c): DBVClose(), DBVsetup_snacc_closerequest()

All these top level functions contains calls to seperate mapping functions for each field in the structure which needs mapping.

Each of these additional mapping functions may themselves have additional subordinate mapping functions.

A new mapping function is always used for mapping each structure in the PDU.

If any memory errors are detected whilst allocating space for Snacc fields (alloc functions return NULL), a longjmp() is made back to the location of the top subordinate call and the error structure passed into the PDU function is completed with diagnostic information. Because of the massive trees of mapping functions, this method is quicker and more efficient that using return values. A similar method is used in snacc generated code for BER-Snacc structure decoding (detailed under ReceiveDataOrigin() and ReceiveDataTarget() descriptions).

Function DBVsetup_snacc_initrequest() from file origin_z3950_init.c is reproduced below together with the first subordinate it calls - DBVset_snacc_initreq_refid() - which maps The DBV InitializeRequest PDU reference if field onto the equivalent Snacc field.

```
int DBVsetup_snacc_initrequest(PDU *PDU_ptr, DbvInitializeReq *psInitializeReq, jmp_buf env)
{

        DBVset_snacc_initreq_refid(PDU_ptr, psInitializeReq->psReferenceId, env);
        DBVset_snacc_initreq_protocolversion(PDU_ptr, psInitializeReq->pcProtocolVersion, env);
        DBVset_snacc_initreq_options(PDU_ptr, psInitializeReq->pcOptions, env);
        DBVset_snacc_initreq_prefmessize(PDU_ptr, psInitializeReq->ulPreferredMessageSize, env);
        DBVset_snacc_initreq_maxmessize(PDU_ptr, psInitializeReq->ulMaximumMessageSize, env);
        DBVset_snacc_initreq_authentication(PDU_ptr, psInitializeReq->psAuthentication, env);
        DBVset_snacc_initreq_impid(PDU_ptr, psInitializeReq->psImplementationId, env);
        DBVset_snacc_initreq_impname(PDU_ptr, psInitializeReq->psImplementationName, env);
        DBVset_snacc_initreq_impver(PDU_ptr, psInitializeReq->psImplementationVersion, env);
        DBVset_snacc_initreq_userinfo(PDU_ptr, psInitializeReq->psUserInformationField, env);
        DBVset_snacc_initreq_otherinfo(PDU_ptr, psInitializeReq->psOtherInformation, env);
}



int DBVset_snacc_initreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
{
        /* Check if a referenceId is given(it is an optional field) */
        if  (psReferenceId != NULL) {
               DBVdbv_to_snacc_octet_string(&(PDU_ptr->a.initRequest->referenceId), psReferenceId,
env);
        }
}
```

Encoding And Sending A Completed Snacc PDU:

After a complete Snacc PDU has been formed with fields corressponding to DBV PDU fields, the Snacc PDU is encoded then sent to the other end of the communication link. Encoding uses automatically generated encoding routines in the z39_* files together with functions and facilities in the snacc run-time library libasn1sbuf.a.

There is only one PDU encoding/send function which is called by all DBV-OSI II PDU functions. This function, DBVencode_and_send() (in z3950_send_receive.c) first calls sub function DBVencode_pdu_to_bytes() (in z3950_enc_dec.) to convert the Snacc PDU structure to a char * containing corressponding 'BER' format data. BER (Basic Encoding Rules) are a set of ISO defined data format rules specifying how to represent an ASN.1 specification in a byte stream. BER was chosen as the low level protocol of Z39.50, hence its specification in ASN.1, but has other completely unrelated uses.

Function DBVencode_pdu_to_bytes() uses a special snacc buffer structure called SBuf (static buffer) which is defined in the snacc run-time libraries. The essentials of this function are shown below:

```
int DBVencode_pdu_to_bytes(DbvAssocId lAssocId, PDU *pdu_ptr, char **encoded_pdu, long
*encoded_pdu_length, DbvError **ppsSIError)
{
.
.
        AsnLen encodedlen;
        static SBuf encodebuffer;
        static char *work_array = NULL;
        int work_array_size = WORK_ARRAY_SIZE;

        /* allocate correct space for work array and change work_array_size as appropriate */
.

        SBufInit(&encodebuffer, work_array, work_array_size);
        SBufResetInWriteRvsMode(&encodebuffer);
        encodedlen = BEncPDU(&encodebuffer, pdu_ptr);
.
.
```

```
        *encoded_pdu = encodebuffer.dataStart;
        *encoded_pdu_length = encodedlen;
.
}
```

Essentially, a completed snacc structure is converted into BER (a char * and a length) by simply calling function BEncPDU! This function has a gigantic structure of called sub functions (in generated files z39_*) but they and their associated complexity are effectively 'hidden' from the application program by the superb snacc interface.

If the Z39.50 specification changes or is updated, the snacc encoding/decoding files can be easily regenerated and the encoding interface in this function remains the same.

Once a char* with a length has been produced, the char * is sent over the communication link. If TCP/IP based communication is being used, this char * is sent by calling function DBVsend_data_over_socket() in file z3950_send_receive.c. If OSI based communication is being used, ISODE Presentation level functions are called to send the char * from within function DBVencode_and_send().

Snacc generates code by default which encodes using 'Definite length' BER encoding rules. This form of encoding is used within DBV-OSI II.

DbvReceiveDataOrigin/ DbvReceiveDataTarget Subordinate Functions

Overview:

The subordinates of DbvReceiveDataOrigin() (in origin_z3950_main.c) and DbvReceiveDataTarget() in (target_z3950_main.c) follow a similar design and structure. As shown in the sections above, there are two main top subordinate functions called by both these functions

1) A function to receive a stream of BER format data for a complete PDU over the network communications and decode it into a Snacc PDU structure (either DBVreceive_data_over_socket() or DBVreceive_data_over_osi() respectively [both functions in file z3950_send_receive.c]).

2) A function to map the Snacc PDU fields produced by the decoding onto similar fields in a DBV PDU structure (either function DBVconvert_pdu_to_origin_data() [in origin_z3950_main.c] for DbvReceiveDataOrigin() or DBVconvert_pdu_to_target_data() [in target_z3950_main.c] for DbvReceiveDataTarget()).

Both of these top subordinate functions is the root of a tree of additional subordinate functions.

Receiveing And Decoding A Snacc PDU:

A complete stream of Z39.50 BER data for a complete PDU is received by either function DBVreceive_data_over_socket() if TCP/IP communication is being used or DBVreceive_data_over_osi() if OSI communication is being used [both functions in file z3950_send_receive.c]).

Both these functions have the same basic format shown below:

```
FUNCTION RECEIVE_AND_DECODE(SNACCPDU *PDU_ptr):

        char *ber_data = NULL;
        unsigned long ber_data_length = 0;
        unsigned long bytes_received = 0;

        /* Until have a complete BER sequence */
        WHILE (!IsCompleteBER(ber_data, ber_data_length))
                bytes_received =RECEIVE_MORE_DATA_OVER_NETWORK();
                ber_data = REALLOC_BER_DATA();
                ber_data_length =  NEW_SIZE_OF_BER_DATA();
                ADD_RECEIVED_DATA_TO_BER_DATA();

        END WHILE

        DECODE_BER_DATA_INTO_SNACCPDU(ber_data, ber_data, length, PDU_ptr);
END FUNCTION.
```

It is necessary to have a loop around data reception code as different targets may send Z39.50 as several packets. The TCP/IP target at z3950.dra.com (port 210) in America for example returns a PDU in 512 byte chunks.

Within DBV-OSI II, a public domain function called IsCompleteBER() developed by OCLC of Ohio, America, is used to determine when a complete PDU has been recieved. IsCompleteBER() is in file berutil.c.

After a complete BER data stream has been received, it is decoded into a SNACC PDU structure by calling function DBVdecode_bytes_to_pdu() in file z3950_enc_dec.c.

This function uses a special snacc buffer structure called SBuf (static buffer) which is defined in the snacc run-time libraries. The essentials of this function are shown below:

```
int DBVdecode_bytes_to_pdu(DbvAssocId lAssocId, char *encoded_pdu, int encoded_pdu_length, PDU
*pdu_ptr, DbvError **ppsSIError)
{

        AsnLen decodedlen;
        SBuf decodebuffer;
        int val;
        jmp_buf env;


        decodedlen = 0;
        SBufInstallData(&decodebuffer, encoded_pdu, encoded_pdu_length);

        /* Do a setjump - snacc 1.1 generated decoding routines do a
           longjmp if decoding fails */
        if ((val = setjmp(env)) == 0) {
                BDecPDU(&decodebuffer, pdu_ptr, &decodedlen, env);
        } else {

                /* Jumps here on decode error */
      }
}
```

Essentially, BER is converted to a completed snacc structure by simply calling function BDecPDU! This function has a gigantic structure of called sub functions (in generated files z39_*) but they and their associated complexity are effectively 'hidden' from the application program by the superb snacc interface.

If an error is detected whilst decoding (such as no memory available for allocing space for a PDU field), a longjmp() call is made which returns execution to the else brace just after the BDecPDU call.

If the Z39.50 specification changes or is updated, the snacc encoding/decoding files can be easily regenerated and the decoding interface in this function remains the same.

Snacc generates code by default which can decode either 'Definite length' or 'Indefinite length' BER data and so caters for all possible Z39.50 BER data formats which could be sent by any target.

Snacc->Dbv Structure Mapping:

After a SNACC PDU has been produced by decoding BER data, a 'Snacc'->DBV PDU field mapping function is called.

DbvReceiveDataOrigin

The top level structure mapping function called by DbvReceiveDataOrigin() is DBVconvert_pdu_to_origin_data() which is defined in file origin_z3950_main.c. This function is reproduced below.

```
int DBVconvert_pdu_to_origin_data(DbvAssocId lAssocId, DbvOriginData *psOriginData, PDU *PDU_ptr,
char *ber_data, unsigned long ber_data_length, jmp_buf env)
{
        int return_value = SI_OK;

        if(DBVTraceLevel(lAssocId) == Trace_High)
                DBVTrace(lAssocId,Trace_High, "\nConverting received PDU to origin data format: ");

        switch (PDU_ptr->choiceId) {
                case PDU_INITRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "InitializeResponse\n");
                        DBVset_dbv_fields_initresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_SEARCHRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "SearchResponse\n");
                        DBVset_dbv_fields_searchresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_PRESENTRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "PresentResponse\n");
                        DBVset_dbv_fields_presresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_DELETERESULTSETRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "DeleteResultSetResponse\n");
                        DBVset_dbv_fields_deleteresultsetresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_RESOURCEREPORTRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "ResourceReportResponse\n");
                        DBVset_dbv_fields_resrepresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_SCANRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "ScanResponse\n");
                        DBVset_dbv_fields_scanresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_SORTRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "SortResponse\n");
                        DBVset_dbv_fields_sortresp(psOriginData, PDU_ptr, env);
```

```
                        break;


                /* Segmentation from server to client is a REQUEST */
                case PDU_SEGMENTREQUEST:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "SegmentRequest\n");
                        DBVset_dbv_fields_segreq(psOriginData, PDU_ptr, env);
                        break;
                case PDU_EXTENDEDSERVICESRESPONSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "ExtendedServicesRequest\n");
                        DBVset_dbv_fields_extendresp(psOriginData, PDU_ptr, env);
                        break;
                case PDU_CLOSE:
                        if(DBVTraceLevel(lAssocId) == Trace_High)
                                DBVTrace(lAssocId,Trace_High, "Close (in Origin)\n");
                        DBVset_dbv_originfields_closereq(psOriginData, PDU_ptr, env);
                        break;

                 case PDU_UNKNOWN:
                          psOriginData->eOriginEvent = OR_Invalid_APDU;
                          psOriginData->uOriginData.psInvalidAPDU = (DbvOctetString *) DbvAlloc
(sizeof(DbvOctetString));
                        CheckDbvAlloc(psOriginData->uOriginData.psInvalidAPDU, env);

                        psOriginData->uOriginData.psInvalidAPDU->pcValue = (char *)
DbvAlloc(ber_data_length);
                        CheckDbvAlloc(psOriginData->uOriginData.psInvalidAPDU->pcValue, env);
                        memcpy(psOriginData->uOriginData.psInvalidAPDU->pcValue, ber_data,
ber_data_length);
                        psOriginData->uOriginData.psInvalidAPDU->ulLength = ber_data_length;
                          /* Now the ber data has been assigned to the DBV
                            structure free it */
                          if (ber_data != NULL)
                                  free(ber_data);
                          break;
                default:
                        break;
        }

        if(return_value == SI_OK && (DBVTraceLevel(lAssocId) == Trace_High))
                DBVTrace(lAssocId,Trace_High, "Conversion successful\n\n");
        return return_value;
}
```

As shown, depending upon the type of the PDU determined (automatically) by the snacc decoding, a DBV_set_fields_<pdu> function is called to map the snacc PDU structure onto an equivalent DBV structure. Each of these Snacc->Dbv structure mapping functions is identical in style and design to the structure of the top level Dbv->Snacc mapping functions EXCEPT THAT THE PDU FIELDS BEING MAPPED ARE REVERSED.

If a PDU of an unknown type is received, the psInvalidAPDU DbvOctetString is filled out with the undecoded received BER data.

Each of the top level setting functions are defined at the end of the file in which they are most logically suited, as listed below:

DBVset_dbv_fields_initresp(): Defined in file origin_z390_init.c

DBVset_dbv_fields_searchresp(): Defined in file origin_z3950_search.c

DBVset_dbv_fields_presresp(): Defined in file origin_z3950_present.c

DBVset_dbv_fields_deleteresultsetresp(): Defined in file origin_z3950_delete.c

DBVset_dbv_fields_resrepresp(): Defined in file origin_z3950_resrep.c

DBVset_dbv_fields_scanresp(): Defined in file origin_z3950_scan.c

DBVset_dbv_fields_sortresp(): Defined in file origin_z3950_sort.c

DBVset_dbv_fields_segreq(): Defined in file origin_z3950_segment.c

DBVset_dbv_fields_extendresp(): Defined in file origin_z3950_extend.c

DBVset_dbv_originfields_closereq(): Defined in file origin_z3950_close.c

DbvReceiveDataTarget

DbvReceiveDataTarget() is almost a mirror image of ReceiveDataOrigin(), with every reference to 'Origin' replaced by 'Target'.

The top level structure mapping function called by DbvReceiveDataTarget() is DBVconvert_pdu_to_target_data() which is defined in file target_z3950_main.c.

Depending upon the type of the PDU determined (automatically) by the snacc decoding, a DBV_set_fields_<pdu> function is called within DBVconvert_pdu_to_target_data() to map the snacc PDU structure onto an equivalent DBV structure. Each of these Snacc->Dbv structure mapping functions is identical in style and design to the structure of the top level Dbv->Snacc mapping functions EXCEPT TH|AT THE PDU FIELDS BEING MAPPED ARE REVERSED.

Each of the top level setting functions are defined at the end of the file in which they are most logically suited, as listed below:

DBVset_dbv_fields_initreq(): Defined in file target_z390_init.c

DBVset_dbv_fields_searchreq(): Defined in file target_z3950_search.c

DBVset_dbv_fields_presreq(): Defined in file target_z3950_present.c

DBVset_dbv_fields_deleteresultsetreq(): Defined in file target_z3950_delete.c

DBVset_dbv_fields_resrepreq(): Defined in file target_z3950_resrep.c

DBVset_dbv_fields_scanreq(): Defined in file origin_z3950_scan.c

DBVset_dbv_fields_sortreq(): Defined in file target_z3950_sort.c

DBVset_dbv_fields_extendreq(): Defined in file target_z3950_extend.c

DBVset_dbv_originfields_closereq(): Defined in file target_z3950_close.c

Complete Subordinate Structure

The complete subordinate structures of all PDU functions in the API, DbvReceiveDataOrigin() and DbvReceiveDataTarget() are reproduced below:

InitializeRequest PDU from origin to target:

origin z3950_init.c:

Function for service user:

```
int DbvInitializeRequest(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvInitReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_initrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psInitializeReq Checking Functions:

```
int DBVCheckDbvInitReqFields(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
int DBVcheck_dbv_initreq_prover()
int DBVcheck_dbv_initreq_options()
int DBVcheck_dbv_initreq_auth()
int DBVcheck_dbv_initreq_userinf()
int DBVcheck_dbv_initreq_otherinfo()

int DBVcheck_dbv_initreq_prover(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_bit_string()
DBVdbv_bit_string_error()

int DBVcheck_dbv_initreq_options(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_bit_string()
DBVcheck_chosen_options()
DBVdbv_options_chosen_error()

int DBVcheck_dbv_initreq_auth(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVcheck_idauth_struct()

int DBVcheck_idauth_struct(DbvAssocId lAssocId, DbvAuthentication *psAuthentication, DbvError
**ppsSIError)
DBVcheck_open(()
DBVcheck_pass()
DBVcheck_idauth_other()
DBVdbv_invalid_enum_error()

int DBVcheck_open(DbvAssocId lAssocId, DbvInternationalString *psOpen, DbvError **ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_pass(DbvAssocId lAssocId, DbvIdPassword *psIdPassword, DbvError **ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_idauth_other(DbvAssocId lAssocId, DbvASN1External *psOther, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_external()

int DBVcheck_dbv_initreq_userinf(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_initreq_otherinfo(DbvAssocId lAssocId, DbvInitializeReq *psInitializeReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psInitializeReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_initrequest(PDU *PDU_ptr, DbvInitializeReq *psInitializeReq, jmp_buf env)
DBVset_snacc_initreq_refid()
DBVset_snacc_initreq_protocolversion()
DBVset_snacc_initreq_options()
DBVset_snacc_initreq_prefmessize()
DBVset_snacc_initreq_maxmessize()
DBVset_snacc_initreq_authentication()
DBVset_snacc_initreq_impid()
DBVset_snacc_initreq_impname()
DBVset_snacc_initreq_impver()
DBVset_snacc_initreq_userinfo()
DBVset_snacc_initreq_otherinfo()

int DBVset_snacc_initreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_initreq_protocolversion(PDU *PDU_ptr, DbvProtocolVersion pcProtocolVersion, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
SetAsnBit()
ClrAsnBit()

int DBVset_snacc_initreq_options(PDU *PDU_ptr, DbvOptions pcOptions, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
SetAsnBit()
ClrAsnBit()

int DBVset_snacc_initreq_prefmessize(PDU *PDU_ptr, Ulong ulPreferredMessageSize, jmp_buf env)

int DBVset_snacc_initreq_maxmessize(PDU *PDU_ptr, Ulong ulMaximumMessageSize, jmp_buf env)

int DBVset_snacc_initreq_authentication(PDU *PDU_ptr, DbvAuthentication *psAuthentication, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVset_snacc_initreq_authentication_open()
DBVset_snacc_initreq_authentication_idpass()
DBVset_snacc_initreq_authentication_other()

int DBVset_snacc_initreq_authentication_open(PDU *PDU_ptr, DbvAuthentication *psAuthentication,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initreq_authentication_idpass(PDU *PDU_ptr, DbvAuthentication *psAuthentication,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initreq_authentication_other(PDU *PDU_ptr, DbvAuthentication *psAuthentication,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_initreq_impid(PDU *PDU_ptr, DbvInternationalString *psImplementationId, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initreq_impname(PDU *PDU_ptr, DbvInternationalString *psImplementationName, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initreq_impver(PDU *PDU_ptr, DbvInternationalString *psImplementationVersion,
jmp_buf env)
DBVdbv_to_snacc_inter_string()
```

```
int DBVset_snacc_initreq_userinfo(PDU *PDU_ptr, DbvASN1External *psUserInformationField, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_initreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target z3950 init.c

Function for service user:

```
int DbvInitializeResponse(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvInitRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_initresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psInitializeRsp Checking Functions:

```
int DBVCheckDbvInitRspFields(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVcheck_dbv_initrsp_prover()
DBVcheck_dbv_initrsp_options()
DBVcheck_dbv_initrsp_result()
DBVcheck_dbv_initrsp_userinf()
DBVcheck_dbv_initrsp_otherinfo()

int DBVcheck_dbv_initrsp_prover(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_bit_string()
DBVdbv_bit_string_error()

int DBVcheck_dbv_initrsp_options(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVdbv_bit_string_error()

int DBVcheck_dbv_initrsp_result(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVcheck_dbv_bool()
DBVdbv_bool_error()

int DBVcheck_dbv_initrsp_userinf(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_initrsp_otherinfo(DbvAssocId lAssocId, DbvInitializeRsp *psInitializeRsp, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psInitializeRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_initresponse(PDU *PDU_ptr, DbvInitializeRsp *psInitializeRsp, jmp_buf env)
DBVset_snacc_initresp_refid()
DBVset_snacc_initresp_protocolversion()
```

```
DBVset_snacc_initresp_options()
DBVset_snacc_initresp_prefmessize()
DBVset_snacc_initresp_maxmessize()
DBVset_snacc_initresp_impid()
DBVset_snacc_initresp_result()
DBVset_snacc_initresp_impname()
DBVset_snacc_initresp_impver()
DBVset_snacc_initresp_userinfo()
DBVset_snacc_initresp_otherinfo()

int DBVset_snacc_initresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_initresp_protocolversion(PDU *PDU_ptr, DbvProtocolVersion pcProtocolVersion,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
SetAsnBit()
ClrAsnBit()

int DBVset_snacc_initresp_options(PDU *PDU_ptr, DbvOptions pcOptions, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
SetAsnBit()
ClrAsnBit()

int DBVset_snacc_initresp_prefmessize(PDU *PDU_ptr, Ulong ulPreferredMessageSize , jmp_buf env)

int DBVset_snacc_initresp_maxmessize(PDU *PDU_ptr, Ulong ulMaximumMessageSize, jmp_buf env)

int DBVset_snacc_initresp_impid(PDU *PDU_ptr, DbvInternationalString *psImplementationId, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initresp_result(PDU *PDU_ptr, DbvBool fResult, jmp_buf env)
int DBVset_snacc_initresp_impname(PDU *PDU_ptr, DbvInternationalString *psImplementationName,
jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initresp_impver(PDU *PDU_ptr, DbvInternationalString *psImplementationVersion,
jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_initresp_userinfo(PDU *PDU_ptr, DbvASN1External *psUserInformationField, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_initresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin z3950_search.c

Function for service user:

```
int DbvSearchRequest(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError ** ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvSearchReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_srchrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSearchReq Checking Functions:

```
int DBVCheckDbvSearchReqFields(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_dbv_srchreq_replace()
DBVcheck_dbv_srchreq_dbnames()
DBVcheck_dbv_srchreq_sselnam()
DBVcheck_dbv_srchreq_mselnam()
DBVcheck_dbv_srchreq_prefsyn()
DBVcheck_dbv_srchreq_query()
DBVcheck_dbv_srchreq_addinf()
DBVcheck_dbv_srchreq_othinfo()

int DBVcheck_dbv_srchreq_replace(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_dbv_bool()
DBVdbv_bool_error()

int DBVcheck_dbv_srchreq_dbnames(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVdbv_field_missing_error()

int DBVcheck_dbv_srchreq_sselnam(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_generic()
DBVcheck_dbspecific()
(DBVdbv_invalid_enum_error()

int DBVcheck_dbv_srchreq_mselnam(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_generic()
DBVcheck_dbspecific()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_srchreq_prefsyn(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_dbv_recsyn()
DBVdbv_oid_error()

int DBVcheck_dbv_srchreq_query(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_rpn()
DBVdbv_invalid_enum_error()

int DBVcheck_rpn(DbvAssocId lAssocId, DbvRPNQuery *psRpnQuery, DbvError **ppsSIError)
DBVcheck_dbv_attsetoid()
DBVdbv_oid_error()
DBVcheck_rpnstruct()

int DBVcheck_rpnstruct(DbvAssocId lAssocId, DbvRPNStructure *psRpnStructure, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_rpnstruct()
DBVcheck_operator()
DBVcheck_operand()
DBVdbv_invalid_enum_error()

int DBVcheck_operator(DbvAssocId lAssocId, DbvOperator *psOperator, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()
DBVcheck_prox()

int DBVcheck_prox(DbvAssocId lAssocId, DbvProximityOperator *psProximityOperator, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_bool()
DBVdbv_bool_error()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_operand(DbvAssocId lAssocId, DbvOperand *psRpnOperand, DbvError **ppsSIError)
DBVdbv_field_missing_error()
```

```
DBVcheck_attsplusterm()
DBVcheck_resplusatts()
DBVdbv_invalid_enum_error()

int DBVcheck_resplusatts(DbvAssocId lAssocId, DbvResultSetPlusAttributes *psResultSetPlusAttributes,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_attelement()

int DBVcheck_dbv_srchreq_addinf(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()

int DBVcheck_dbv_srchreq_othinfo(DbvAssocId lAssocId, DbvSearchReq *psSearchReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psSearchReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_srchrequest(PDU *PDU_ptr, DbvSearchReq *psSearchReq, jmp_buf env)
DBVset_snacc_searchreq_refid()
DBVset_snacc_searchreq_ssetupper()
DBVset_snacc_searchreq_lsetlower()
DBVset_snacc_searchreq_msetpres()
DBVset_snacc_searchreq_replace()
DBVset_snacc_searchreq_resultset()
DBVset_snacc_searchreq_dbnames()
DBVset_snacc_searchreq_ssetelname()
DBVset_snacc_searchreq_msetelname()
DBVset_snacc_searchreq_prefrecsyn()
DBVset_snacc_searchreq_query()
DBVset_snacc_searchreq_addinfo()
DBVset_snacc_searchreq_otherinfo()

int DBVset_snacc_searchreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_searchreq_ssetupper(PDU *PDU_ptr,  Ulong ulSmallSetUpperBound, jmp_buf env)

int DBVset_snacc_searchreq_lsetlower(PDU *PDU_ptr, Ulong ulLargeSetLowerBound, jmp_buf env)

int DBVset_snacc_searchreq_msetpres(PDU *PDU_ptr, Ulong ulMediumSetPresentNumber, jmp_buf env)

int DBVset_snacc_searchreq_replace(PDU *PDU_ptr, DbvBool fReplaceIndicator, jmp_buf env)

int DBVset_snacc_searchreq_resultset(PDU *PDU_ptr, DbvInternationalString sResultSetName, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_searchreq_dbnames(PDU *PDU_ptr, DbvInternationalString **ppsDatabaseNames, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_searchreq_ssetelname(PDU *PDU_ptr, DbvElementSetNames *psSmallSetElementNames,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_el_set_names()

int DBVset_snacc_searchreq_msetelname(PDU *PDU_ptr, DbvElementSetNames *psMediumSetElementNames,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_el_set_names()
```

```
int DBVset_snacc_searchreq_prefrecsyn(PDU *PDU_ptr, DbvObjectIdentifier pcPreferredRecordSyntax,
jmp_buf env)
ASN_MakeOid()
```

oid tool.c:

```
void ASN_MakeOid(char *dottedString, AsnOid * oidEntry, jmp_buf env)
RescIR_Zmalloc()
CheckAsn1Alloc()
EncodedOidLen()
ASN_OctsCreate()
BuildEncodedOid()
ASN_FreeOid()


char *RescIR_Zmalloc(FILE * f, int size)

int ASN_OctsCreate(char *tmpBuf, AsnLen oidByteLen, AsnOid * oidEntry, jmp_buf env)
CheckAsn1Alloc()

void ASN_FreeOid(OID_SNACC * oidList)
RescIR_Free()

void RescIR_Free(FILE * f, char *str)
```

origin z3950_search.c:

```
int DBVset_snacc_searchreq_query(PDU *PDU_ptr, DbvQuery *psQuery, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_octet_string()
DBVdbv_to_snacc_rpnquery()

int DBVdbv_to_snacc_rpnquery(struct RPNQuery* type_1, DbvRPNQuery *psRpnQuery, jmp_buf env)
ASN_MakeOid()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_rpnstruct()

int DBVdbv_to_snacc_rpnstruct(struct RPNStructure* rpn, DbvRPNStructure *psRpnStructure, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_rpnrpnop()
DBVdbv_to_snacc_operand()

int DBVdbv_to_snacc_rpnrpnop(struct RPNStructureSeq* rpnRpnOp, DbvRPNLopRopOperator
*psRpnLopRopOperator, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_rpnstruct()
DBVdbv_to_snacc_operator()

int DBVdbv_to_snacc_operator(struct Operator* op, DbvOperator sOperator, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_prox()

int DBVdbv_to_snacc_prox(struct ProximityOperator* prox, DbvProximityOperator *psProximityOperator,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_proxunitcode()

int DBVdbv_to_snacc_proxunitcode(struct ProximityOperatorChoice* proximityUnitCode,
DbvProximityUnitCode sProximityUnitCode, jmp_buf env)

int DBVdbv_to_snacc_operand(struct Operand* op, DbvOperand* psRpnOperand, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
```

```
DBVdbv_to_snacc_attrterm()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_resulattr()

int DBVdbv_to_snacc_resulattr(struct ResultSetPlusAttributes* resultAttr, DbvResultSetPlusAttributes
*psResultSetPlusAttributes, jmp_buf env)
DBVdbv_to_snacc_inter_string()
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_attribute_list()

int DBVset_snacc_searchreq_addinfo(PDU *PDU_ptr, DbvOtherInformation *psAdditionalSearchInfo,
jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()

int DBVset_snacc_searchreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_search.c

Function for service user:

```
int DbvSearchResponse(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError * *ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvSearchRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_srchresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSearchReq Checking Functions:

```
int DBVCheckDbvSearchRspFields(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVcheck_dbv_srchrsp_sstat()
DBVcheck_dbv_srchrsp_ressetst()
DBVcheck_dbv_srchrsp_presst()
DBVcheck_dbv_srchrsp_records()
DBVcheck_dbv_srchrsp_addinf()
DBVcheck_dbv_srchrsp_othinfo()

int DBVcheck_dbv_srchrsp_sstat(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVdbv_bool_error()

int DBVcheck_dbv_srchrsp_ressetst(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_srchrsp_presst(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_srchrsp_records(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVcheck_dbv_records()

int DBVcheck_dbv_srchrsp_addinf(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
```

```
DBVcheck_dbv_otherinfo()

int DBVcheck_dbv_srchrsp_othinfo(DbvAssocId lAssocId, DbvSearchRsp *psSearchRsp, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psSearchRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_srchresponse(PDU *PDU_ptr, DbvSearchRsp *psSearchRsp, jmp_buf env)
DBVset_snacc_searchresp_refid()
DBVset_snacc_searchresp_rescount()
DBVset_snacc_searchresp_numrecs()
DBVset_snacc_searchresp_nextres()
DBVset_snacc_searchresp_srchstat()
DBVset_snacc_searchresp_resstat()
DBVset_snacc_searchresp_presstat()
DBVset_snacc_searchresp_records()
DBVset_snacc_searchresp_addinfo()
DBVset_snacc_searchresp_otherinfo()

int DBVset_snacc_searchresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_searchresp_rescount(PDU *PDU_ptr, Ulong ulResultCount, jmp_buf env)

int DBVset_snacc_searchresp_numrecs(PDU *PDU_ptr, Ulong ulNumberOfRecordsReturned, jmp_buf env)

int DBVset_snacc_searchresp_nextres(PDU *PDU_ptr, Ulong ulNextResultSetPosition, jmp_buf env)

int DBVset_snacc_searchresp_srchstat(PDU *PDU_ptr, DbvBool fSearchStatus, jmp_buf env)

int DBVset_snacc_searchresp_resstat(PDU *PDU_ptr, DbvResultSetStatus *peResultSetStatus, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_searchresp_presstat(PDU *PDU_ptr, DbvPresentStatus *pePresentStatus, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_searchresp_records(PDU *PDU_ptr, DbvRecords *psRecords, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_records()

int DBVset_snacc_searchresp_addinfo(PDU *PDU_ptr, DbvOtherInformation *psAdditionalSearchInfo,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()

int DBVset_snacc_searchresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin_z3950_present.c

Function for service user:

```
int DbvPresentRequest(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvPresReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_presrequest()
DBVfill_error_structure()
```

```
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSearchReq Checking Functions:

```
int DBVCheckDbvPresReqFields(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError
**ppsSIError)
DBVcheck_dbv_presreq_addrang()
DBVcheck_dbv_presreq_reccomp()
DBVcheck_dbv_presreq_prefsyn()
DBVcheck_dbv_presreq_othinfo()

int DBVcheck_dbv_presreq_addrang(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_dbv_presreq_reccomp(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError
**ppsSIError)
DBVcheck_el_set_names()
DBVcheck_compspec()
DBVdbv_invalid_enum_error()

int DBVcheck_el_set_names(DbvAssocId lAssocId, DbvElementSetNames *psElementSetNames, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_generic()
DBVcheck_dbspecific()
DBVdbv_invalid_enum_error()

int DBVcheck_compspec(DbvAssocId lAssocId, DbvCompSpec *psCompSpec, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_bool()
DBVdbv_bool_error()
DBVcheck_specification()
DBVcheck_spec_comp()
DBVcheck_obj_list()

int DBVcheck_spec_comp(DbvAssocId lAssocId, DbvDbSpecificComp **ppsDbSpecific, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_specification()

int DBVcheck_obj_list(DbvAssocId lAssocId, DbvObjectIdentifier *ppcRecordSyntax, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_recsyn()
DBVdbv_oid_error()

int DBVcheck_dbv_presreq_prefsyn(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError
**ppsSIError)
DBVcheck_dbv_recsyn()
DBVdbv_field_missing_error()

int DBVcheck_dbv_presreq_othinfo(DbvAssocId lAssocId, DbvPresentReq *psPresentReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psPresentReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_presrequest(PDU *PDU_ptr, DbvPresentReq *psPresentReq, jmp_buf env)
DBVset_snacc_presreq_refid()
DBVset_snacc_presreq_ressetid()
DBVset_snacc_presreq_ressetstart()
DBVset_snacc_presreq_numrecsreq()
DBVset_snacc_presreq_addranges()
DBVset_snacc_presreq_reccomp()
DBVset_snacc_presreq_prefrecsyn()
DBVset_snacc_presreq_maxsegcnt()
DBVset_snacc_presreq_maxrecsze()
DBVset_snacc_presreq_maxsegsze()
DBVset_snacc_presreq_otherinfo()
```

```
int DBVset_snacc_presreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_presreq_ressetid(PDU *PDU_ptr, DbvInternationalString sResultSetId, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_presreq_ressetstart(PDU *PDU_ptr, Ulong ulResultSetStartPoint, jmp_buf env)

int DBVset_snacc_presreq_numrecsreq(PDU *PDU_ptr, Ulong ulNumberOfRecordsRequested, jmp_buf env)

int DBVset_snacc_presreq_addranges(PDU *PDU_ptr, DbvRange **ppsAdditionalRanges, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()

int DBVset_snacc_presreq_reccomp(PDU *PDU_ptr, DbvRecordComposition *psRecordComposition, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_el_set_names()
DBVdbv_to_snacc_presreq_complex()

int DBVdbv_to_snacc_presreq_complex(struct CompSpec* complex, DbvCompSpec *psCompSpec, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_specification()
AsnListNew()
DBVdbv_to_snacc_pres_dbspecific()
DBVdbv_to_snacc_pres_recsyntax()

int DBVdbv_to_snacc_pres_dbspecific(CompSpecSeqOf *dbSpecific, DbvDbSpecificComp **ppsDbSpecific,
jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_specification()

int DBVdbv_to_snacc_pres_recsyntax(CompSpecSeqOf1* recordSyntax,  DbvObjectIdentifier
*ppcRecordSyntax, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
ASN_MakeOid()

int DBVset_snacc_presreq_prefrecsyn(PDU *PDU_ptr, DbvObjectIdentifier pcPreferredRecordSyntax,
jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_presreq_maxsegcnt(PDU *PDU_ptr, Ulong *pulMaxSegmentCount, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_presreq_maxrecsze(PDU *PDU_ptr, Ulong *pulMaxRecordSize, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_presreq_maxsegsze(PDU *PDU_ptr, Ulong *pulMaxSegmentSize, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_presreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_present.c

Function for service user:

```
int DbvPresentResponse(DbvAssocId lAssocId, DbvPresentRsp *psPresentRsp, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvPresRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_presresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psPresentRsp Checking Functions:

```
int DBVCheckDbvPresRspFields(DbvAssocId lAssocId, DbvPresentRsp *psPresentRsp, DbvError
**ppsSIError)
DBVcheck_dbv_presrsp_pstat()
DBVcheck_dbv_presrsp_records()
DBVcheck_dbv_presrsp_othinfo()

int DBVcheck_dbv_presrsp_pstat(DbvAssocId lAssocId, DbvPresentRsp *psPresentRsp, DbvError
**ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_presrsp_records(DbvAssocId lAssocId, DbvPresentRsp *psPresentRsp, DbvError
**ppsSIError)
DBVcheck_dbv_records()

int DBVcheck_dbv_presrsp_othinfo(DbvAssocId lAssocId, DbvPresentRsp *psPresentRsp, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psPresentRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_presresponse(PDU *PDU_ptr, DbvPresentRsp *psPresentRsp, jmp_buf env)
DBVset_snacc_presresp_refid()
DBVset_snacc_presresp_numrecs()
DBVset_snacc_presresp_nxtresset()
DBVset_snacc_presresp_presstat()
DBVset_snacc_presresp_records()
DBVset_snacc_presresp_otherinfo()

int DBVset_snacc_presresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_presresp_numrecs(PDU *PDU_ptr, Ulong ulNumberOfRecordsReturned, jmp_buf env)

int DBVset_snacc_presresp_nxtresset(PDU *PDU_ptr, Ulong ulNextResultSetPosition, jmp_buf env)

int DBVset_snacc_presresp_presstat(PDU *PDU_ptr, DbvPresentStatus ePresentStatus, jmp_buf env)

int DBVset_snacc_presresp_records(PDU *PDU_ptr, DbvRecords *psRecords, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_records()

int DBVset_snacc_presresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin z3950_delete.c

Function for service user:

```
int DbvDeleteResultSetRequest(DbvAssocId lAssocId, DbvDeleteResultSetReq *psDeleteResultSetReq,
DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvDelReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_delrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psDeleteReq Checking Functions:

```
int DBVCheckDbvDelReqFields(DbvAssocId lAssocId, DbvDeleteResultSetReq *psDeleteResultSetReq,
DbvError **ppsSIError)
DBVcheck_dbv_delreq_delfunc()
DBVcheck_dbv_delreq_resset()
DBVcheck_dbv_delreq_otherinfo()

int DBVcheck_dbv_delreq_delfunc(DbvAssocId lAssocId, DbvDeleteResultSetReq *psDeleteResultSetReq,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_delreq_resset(DbvAssocId lAssocId, DbvDeleteResultSetReq *psDeleteResultSetReq,
DbvError **ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_dbv_delreq_otherinfo(DbvAssocId lAssocId, DbvDeleteResultSetReq *psDeleteResultSetReq,
DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psDeletResultSetReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_delrequest(PDU *PDU_ptr, DbvDeleteResultSetReq *psDeleteResultSetReq, jmp_buf
env)
DBVset_snacc_delreq_refid()
DBVset_snacc_delreq_deletefunc()
DBVset_snacc_delreq_ressetid()
DBVset_snacc_delreq_otherinfo()

int DBVset_snacc_delreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_delreq_deletefunc(PDU *PDU_ptr, DbvDeleteFunction eDeleteFunction, jmp_buf env)

int DBVset_snacc_delreq_ressetid(PDU *PDU_ptr, DbvInternationalString **ppsResultSetId, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_delreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target z3950 delete.c

Function for service user:

```
int DbvDeleteResultSetResponse(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp,
DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDelRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_delresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psDeleteRsp Checking Functions:

```
int DBVCheckDelRspFields(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp, DbvError
**ppsSIError)
DBVcheck_dbv_delrsp_opstat()
DBVcheck_dbv_delrsp_liststat()
DBVcheck_dbv_delrsp_bulkstat()
DBVcheck_dbv_delrsp_otherinfo()

int DBVcheck_dbv_delrsp_opstat(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_delrsp_liststat(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_liststat()

int DBVcheck_liststat(DbvAssocId lAssocId, char *field, DbvListStatus **ppsDeleteListStatuses,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_delrsp_bulkstat(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_liststat()

int DBVcheck_dbv_delrsp_otherinfo(DbvAssocId lAssocId, DbvDeleteResultSetRsp *psDeleteResultSetRsp,
DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psDeletResultSetRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_delresponse(PDU *PDU_ptr, DbvDeleteResultSetRsp *psDeleteResultSetRsp, jmp_buf
env)
int DBVset_snacc_delresp_refid()
int DBVset_snacc_delresp_opstat()
int DBVdbv_to_snacc_delstat()
int DBVset_snacc_delresp_delliststat()
int DBVset_snacc_delresp_numnotdel()
int DBVset_snacc_delresp_bulkliststat()
int DBVset_snacc_delresp_delmess()
int DBVset_snacc_delresp_otherinfo()

int DBVset_snacc_delresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_delresp_opstat(PDU *PDU_ptr, DbvDeleteSetStatus eDeleteOperationStatus, jmp_buf
env)
DBVdbv_to_snacc_delstat()

int DBVdbv_to_snacc_delstat(DeleteSetStatus *snacc_status, DbvDeleteSetStatus *dbv_status, jmp_buf
env)
```

```
int DBVset_snacc_delresp_delliststat(PDU *PDU_ptr, DbvListStatus **ppsDeleteListStatuses, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_liststat()

int DBVdbv_to_snacc_liststat(ListStatusesSeq *data_ptr, DbvListStatus *psDeleteListStatuses, jmp_buf
env)
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_delstat()

int DBVset_snacc_delresp_numnotdel(PDU *PDU_ptr, Ulong *pulNumberNotDeleted, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_delresp_bulkliststat(PDU *PDU_ptr, DbvListStatus **ppsBulkStatuses, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_liststat()

int DBVset_snacc_delresp_delmess(PDU *PDU_ptr, DbvInternationalString *psDeleteMessage, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_delresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin z3950 sort.c

Function for service user:

```
int DbvSortRequest(DbvAssocId lAssocId, DbvSortReq *psSortReq, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvSortReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_sortrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSortReq Checking Functions:

```
int DBVCheckDbvSortReqFields(DbvAssocId lAssocId, DbvSortReq *psSortReq, DbvError **ppsSIError)
DBVcheck_dbv_sortreq_inresname()
DBVcheck_dbv_sortreq_sequence()
DBVcheck_dbv_sortreq_otherinfo()

int DBVcheck_dbv_sortreq_inresname(DbvAssocId lAssocId, DbvSortReq *psSortReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_dbv_sortreq_sequence(DbvAssocId lAssocId, DbvSortReq *psSortReq, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_sortkeyspec()

int DBVcheck_sortkeyspec(DbvAssocId lAssocId, DbvSortKeySpec *psSortSequence, char *field, DbvError
**ppsSIError)
DBVcheck_sortelmnt()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()
DBVcheck_sortmvaction()
```

```
int DBVcheck_sortelmnt(DbvAssocId lAssocId, DbvSortElement *psSortElement, char *field, DbvError
**ppsSIError)
DBVcheck_sortkey()
DBVcheck_dbspecsortkey()
DBVdbv_invalid_enum_error()

int DBVcheck_sortkey(DbvAssocId lAssocId, DbvSortKey *psGeneric, char *field, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_specification()
DBVcheck_sortatts()
DBVdbv_invalid_enum_error()

int DBVcheck_sortatts(DbvAssocId lAssocId, DbvSortAttributes *psSortAttributes, char *field,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_sortattsetid()
DBVcheck_sortattelmnt()

int DBVcheck_sortattsetid(DbvAssocId lAssocId, DbvObjectIdentifier pcAttributeSetId, char *field,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_attsetoid()
DBVdbv_oid_error()

int DBVcheck_sortattelmnt(DbvAssocId lAssocId, DbvAttributeElement **ppsAttributeElement, char
*field, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVdbv_field_missing_error()
DBVcheck_attelement()

int DBVcheck_dbspecsortkey(DbvAssocId lAssocId, DbvDbSpecificSortKey **ppsDatabaseSpecific, char
*field, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_specsortkey()

int DBVcheck_specsortkey(DbvAssocId lAssocId, DbvDbSpecificSortKey *psDatabaseSpecific, char *field,
DbvError **ppsSIError)
DBVcheck_sortkey()

int DBVcheck_sortmvaction(DbvAssocId lAssocId, DbvSortMVAction *psMVActionType, char *field,
DbvError **ppsSIError)
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_sortreq_otherinfo(DbvAssocId lAssocId, DbvSortReq *psSortReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psSortReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_sortrequest(PDU *PDU_ptr, DbvSortReq *psSortReq, jmp_buf env)
DBVset_snacc_sortreq_refid()
DBVset_snacc_sortreq_inresset()
DBVset_snacc_sortreq_sortedres()
DBVset_snacc_sortreq_sortseq()
DBVset_snacc_sortreq_otherinfo()

int DBVset_snacc_sortreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_sortreq_inresset(PDU *PDU_ptr, DbvInternationalString **ppsInputResultSetName,
jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_sortreq_sortedres(PDU *PDU_ptr, DbvInternationalString sSortedResultSetName,
jmp_buf env)
DBVdbv_to_snacc_inter_string()
```

```
int DBVset_snacc_sortreq_sortseq(PDU *PDU_ptr, DbvSortKeySpec **ppsSortSequence, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_sortkeyspec()

int DBVdbv_to_snacc_sortkeyspec(SortKeySpec *data_ptr, DbvSortKeySpec *psSortSequence, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_sortelmnt()
DBVdbv_to_snacc_sortrel()
DBVdbv_to_snacc_sortcase()
DBVdbv_to_snacc_sortmv()

int DBVdbv_to_snacc_sortelmnt(struct SortElement* sortElement, DbvSortElement *psSortElement,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_sortkey()
AsnListNew()
DBVdbv_to_snacc_dbspecific()

int DBVdbv_to_snacc_sortkey(struct SortKey* generic, DbvSortKey *psGeneric, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_specification()
DBVdbv_to_snacc_sortatts()

int DBVdbv_to_snacc_sortatts(struct SortKeySeq* sortAttributes, DbvSortAttributes *psSortAttributes,
jmp_buf env)
ASN_MakeOid()
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_attribute_list()

int DBVdbv_to_snacc_dbspecific(SortElementSeqOf *datbaseSpecific, DbvDbSpecificSortKey
**ppsDatabaseSpecific, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_specsortkey()

int DBVdbv_to_snacc_specsortkey(SortElementSeqOfSeq *data_ptr, DbvDbSpecificSortKey
*psDatabaseSpecific, jmp_buf env)
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_sortkey()

int DBVdbv_to_snacc_sortrel(SortKeySpec *data_ptr, DbvSortKeySpec *psSortSequence, jmp_buf env)

int DBVdbv_to_snacc_sortcase(SortKeySpec *data_ptr, DbvSortKeySpec *psSortSequence, jmp_buf env)

int DBVdbv_to_snacc_sortmv(struct SortKeySpecChoice* missingValueAction, DbvSortMVAction
*psMVActionType, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_sortreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_sort.c

Function for service user:

```
int DbvSortResponse(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvSortRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_sortresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSortRsp Checking Functions:

```
int DBVCheckDbvSortRspFields(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError **ppsSIError)
DBVcheck_dbv_sortrsp_status()
DBVcheck_dbv_sortrsp_ressetid()
DBVcheck_dbv_sortrsp_diag()
DBVcheck_dbv_sortrsp_otherinfo()

int DBVcheck_dbv_sortrsp_status(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_sortrsp_ressetid(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_sortrsp_diag(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_multdiag()

int DBVcheck_dbv_sortrsp_otherinfo(DbvAssocId lAssocId, DbvSortRsp *psSortRsp, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psSortRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_sortresponse(PDU *PDU_ptr, DbvSortRsp *psSortRsp, jmp_buf env)
DBVset_snacc_sortresp_refid()
DBVset_snacc_sortresp_sortstat()
DBVset_snacc_sortresp_resstat()
DBVset_snacc_sortresp_diag()
DBVset_snacc_sortresp_otherinfo()

int DBVset_snacc_sortresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_sortresp_sortstat(PDU *PDU_ptr, DbvSortStatus eSortStatus, jmp_buf env)

int DBVset_snacc_sortresp_resstat(PDU *PDU_ptr, DbvSortResultSetStatus *peResultSetStatus, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_sortresp_diag(PDU *PDU_ptr, DbvDiagRec **ppsDiagnostics, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_multdiag()

int DBVset_snacc_sortresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin_z3950_scan.c:

Function for service user:

```
int DbvScanRequest(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvScanReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_scanrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psScanReq Checking Functions:

```
int DBVCheckDbvScanReqFields(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError **ppsSIError)
DBVcheck_dbv_scanreq_dbnames()
DBVcheck_dbv_scanreq_attset()
DBVcheck_dbv_scanreq_atttrm()
DBVcheck_dbv_scanreq_otherinfo()

int DBVcheck_dbv_scanreq_dbnames(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError **ppsSIError)
DBVdbv_field_missing_error()

int DBVcheck_dbv_scanreq_attset(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError **ppsSIError)
DBVcheck_dbv_attsetoid()
DBVdbv_oid_error()

int DBVcheck_dbv_scanreq_atttrm(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError **ppsSIError)
DBVcheck_attsplusterm()

int DBVcheck_dbv_scanreq_otherinfo(DbvAssocId lAssocId, DbvScanReq *psScanReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psScanReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_scanrequest(PDU *PDU_ptr, DbvScanReq *psScanReq, jmp_buf env)
DBVset_snacc_scanreq_refid()
DBVset_snacc_scanreq_dbnames()
DBVset_snacc_scanreq_attsetid()
DBVset_snacc_scanreq_attplusterm()
DBVset_snacc_scanreq_stepsize()
DBVset_snacc_scanreq_numterms()
DBVset_snacc_scanreq_prefpos()
DBVset_snacc_scanreq_otherinfo()

int DBVset_snacc_scanreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_scanreq_dbnames(PDU *PDU_ptr, DbvInternationalString **ppsDatabaseNames, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_scanreq_attsetid(PDU *PDU_ptr, DbvObjectIdentifier pcAttributeSetId, jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_scanreq_attplusterm(PDU *PDU_ptr, DbvAttributesPlusTerm sAttributesPlusTerm,
jmp_buf env)
```

```
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_attrterm()

int DBVset_snacc_scanreq_stepsize(PDU *PDU_ptr, Ulong *pulStepSize, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_scanreq_numterms(PDU *PDU_ptr, Ulong ulNumberOfTermsRequested, jmp_buf env)

int DBVset_snacc_scanreq_prefpos(PDU *PDU_ptr, Ulong *pulPreferredPositionInResponse, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_scanreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_scan.c:

Function for service user:

```
int DbvScanResponse(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvScanRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_scanresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psScanRsp Checking Functions:

```
int DBVCheckDbvScanRspFields(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError **ppsSIError)
DBVcheck_dbv_scanrsp_stat()
DBVcheck_dbv_scanrsp_entries()
DBVcheck_dbv_scanrsp_attset()
DBVcheck_dbv_scanrsp_otherinfo()

int DBVcheck_dbv_scanrsp_stat(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_scanrsp_entries(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError **ppsSIError)
DBVcheck_dbv_entries()

int DBVcheck_dbv_entries(DbvAssocId lAssocId, DbvListEntries *psEntries, DbvError **ppsSIError)
DBVcheck_entry_list()
DBVdbv_field_missing_error()
DBVcheck_dbv_multdiag()
DBVdbv_invalid_enum_error()

int DBVcheck_entry_list(DbvAssocId lAssocId, DbvEntry **ppsEntry, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_entry()

int DBVcheck_entry(DbvAssocId lAssocId, DbvEntry *psEntry, char *error_buffer, DbvError
**ppsSIError)
DBVcheck_terminfo()
DBVdbv_field_missing_error()
DBVcheck_dbv_surrdiag()
DBVdbv_invalid_enum_error()
```

```
int DBVcheck_terminfo(DbvAssocId lAssocId, DbvTermInfo *psTermInfo, char* field, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_term()
DBVcheck_attel_list()
DBVcheck_attsplusterm_list()
DBVcheck_byattributes()
DBVcheck_dbv_otherinfo()

int DBVcheck_attel_list(DbvAssocId lAssocId, DbvAttributeElement **ppsAttributeElement, char *field,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_attelement()

int DBVcheck_attsplusterm_list(DbvAssocId lAssocId, DbvAttributesPlusTerm **ppsAlternativeTerm, char
*field, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_attsplusterm()

int DBVcheck_byattributes(DbvAssocId lAssocId, DbvOccurrenceByAttributes *psByAttributes, char
*field, DbvError **ppsSIError)
DBVcheck_attel_list_comp()
DBVcheck_occurrence()
DBVcheck_dbv_otherinfo()

int DBVcheck_attel_list_comp(DbvAssocId lAssocId, DbvAttributeElement **ppsAttributeElement, char
*field, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_attelement()

int DBVcheck_occurrence(DbvAssocId lAssocId, DbvOccurrence *psOccurrence, char *field, DbvError
**ppsSIError)
DBVcheck_occurbydb_list()
DBVdbv_invalid_enum_error()

int DBVcheck_occurbydb_list(DbvAssocId lAssocId, DbvOccurrenceByDb **ppsOccurrenceByDb, char *field,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_occurbydb()

int DBVcheck_occurbydb(DbvAssocId lAssocId, DbvOccurrenceByDb *psOccurrenceByDb, char *field,
DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()

int DBVcheck_dbv_scanrsp_attset(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError **ppsSIError)
DBVcheck_dbv_attsetoid()
DBVdbv_oid_error()

int DBVcheck_dbv_scanrsp_otherinfo(DbvAssocId lAssocId, DbvScanRsp *psScanRsp, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psScanRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_scanresponse(PDU *PDU_ptr, DbvScanRsp *psScanRsp, jmp_buf env)
DBVset_snacc_scanresp_refid()
DBVset_snacc_scanresp_stepsize()
DBVset_snacc_scanresp_scanstat()
DBVset_snacc_scanresp_numentries()
DBVset_snacc_scanresp_posterm()
DBVset_snacc_scanresp_entries()
DBVset_snacc_scanresp_attsetid()
DBVset_snacc_scanresp_otherinfo()

int DBVset_snacc_scanresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_scanresp_stepsize(PDU *PDU_ptr, Ulong *pulStepSize, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
int DBVset_snacc_scanresp_scanstat(PDU *PDU_ptr, DbvScanStatus eScanStatus, jmp_buf env)
```

```
int DBVset_snacc_scanresp_numentries(PDU *PDU_ptr,Ulong ulNumberOfEntriesReturned, jmp_buf env)

int DBVset_snacc_scanresp_posterm(PDU *PDU_ptr, Ulong *pulPositionOfTerm, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVset_snacc_scanresp_entries(PDU *PDU_ptr, DbvListEntries *psEntries, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_listent()

int DBVdbv_to_snacc_listent(struct ListEntries *entries, DbvListEntries *psEntries, jmp_buf env
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_entries()
DBVdbv_to_snacc_multdiag()

int DBVdbv_to_snacc_entries(ListEntriesSeqOf *entries, DbvEntry **ppsEntry, jmp_buf env)
AsnListAppend
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_entry()

int DBVdbv_to_snacc_entry(Entry *data_ptr, DbvEntry *psEntry, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_terminfo()
DBVdbv_to_snacc_diag_rec()

int DBVdbv_to_snacc_terminfo(struct TermInfo* termInfo, DbvTermInfo *psTermInfo, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_term()
AsnListNew()
DBVdbv_to_snacc_attribute_list()
DBVdbv_to_snacc_attrtermlist()
DBVdbv_to_snacc_occurbyatt()
DBVdbv_to_snacc_otherinfo()

int DBVdbv_to_snacc_attrtermlist(TermInfoSeqOf *alternativeTerm, DbvAttributesPlusTerm
**ppsAlternativeTerm, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_attrterm()

int DBVdbv_to_snacc_occurbyatt(OccurrenceByAttributes *byAttributes, DbvOccurrenceByAttributes
*psByAttributes, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_occurbyattseq()

int DBVdbv_to_snacc_occurbyattseq(OccurrenceByAttributesSeq *data_ptr, DbvOccurrenceByAttributes
*psByAttributes, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_attribute_list()
Asn1Alloc()
DVBVdbv_to_snacc_occurrence()
DBVdbv_to_snacc_otherinfo()

int DVBVdbv_to_snacc_occurrence(struct OccurrenceByAttributesSeqChoice* occurrences, DbvOccurrence
*psOccurrence, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_bydatabase()

int DBVdbv_to_snacc_bydatabase(OccurrenceByAttributesSeqChoiceSeqOf *byDatabase, DbvOccurrenceByDb
**ppsOccurrenceByDb, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
```

```
DBVdbv_to_snacc_bigoccurrthing()

int DBVdbv_to_snacc_bigoccurrthing(OccurrenceByAttributesSeqChoiceSeqOfSeq *data_ptr,
DbvOccurrenceByDb *psOccurrenceByDb, jmp_buf env)
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()
AsnListNew()
DBVdbv_to_snacc_otherinfo()

int DBVset_snacc_scanresp_attsetid(PDU *PDU_ptr, DbvObjectIdentifier pcAttributeSetId, jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_scanresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

origin_z3950_resrep.c:

Function for service user:

```
int DbvResourceReportRequest(DbvAssocId lAssocId, DbvResourceReportReq *psResourceReportReq,
DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvResRepReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_resreprequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psResourceReportReq Checking Functions:

```
int DBVCheckDbvResRepReqFields(DbvAssocId lAssocId, DbvResourceReportReq *psResourceReportReq,
DbvError **ppsSIError)
DBVcheck_dbv_rerereq_preffmt()
DBVcheck_dbv_rerereq_otherinfo()

int DBVcheck_dbv_rerereq_preffmt(DbvAssocId lAssocId, DbvResourceReportReq *psResourceReportReq,
DbvError **ppsSIError)
DBVcheck_dbv_resrepoid()
DBVdbv_oid_error()

int DBVcheck_dbv_rerereq_otherinfo(DbvAssocId lAssocId, DbvResourceReportReq *psResourceReportReq,
DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psResourceReportReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_resreprequest(PDU *PDU_ptr, DbvResourceReportReq *psResourceReportReq, jmp_buf
env)
DBVset_snacc_resrepreq_refid()
DBVset_snacc_resrepreq_opid()
DBVset_snacc_resrepreq_prefformat()
DBVset_snacc_resrepreq_otherinfo()

int DBVset_snacc_resrepreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_resrepreq_opid(PDU *PDU_ptr, DbvOctetString *psOperationId, jmp_buf env)
```

```
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_resrepreq_prefformat(PDU *PDU_ptr, DbvObjectIdentifier pcPrefResourceReportFormat,
jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_resrepreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target z3950 resrep.c:

Function for service user:

```
int DbvResourceReportResponse(DbvAssocId lAssocId, DbvResourceReportRsp *psResourceReportRsp,
DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvResRepRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_resrepresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psResourceReportRsp Checking Functions:

```
int DBVCheckDbvResRepRspFields(DbvAssocId lAssocId, DbvResourceReportRsp *psResourceReportRsp,
DbvError **ppsSIError)
DBVcheck_dbv_rerersp_repstat()
DBVcheck_dbv_rerersp_resrep()
DBVcheck_dbv_rerersp_otherinfo()

int DBVcheck_dbv_rerersp_repstat(DbvAssocId lAssocId, DbvResourceReportRsp *psResourceReportRsp,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_rerersp_resrep(DbvAssocId lAssocId, DbvResourceReportRsp *psResourceReportRsp,
DbvError **ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_rerersp_otherinfo(DbvAssocId lAssocId, DbvResourceReportRsp *psResourceReportRsp,
DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psResourceReportRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_resrepresponse(PDU *PDU_ptr, DbvResourceReportRsp *psResourceReportRsp, jmp_buf
env)
DBVset_snacc_resrepresp_refid()
DBVset_snacc_resrepresp_repstat()
DBVset_snacc_resrepresp_report()
DBVset_snacc_resrepresp_otherinfo()

int DBVset_snacc_resrepresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_resrepresp_repstat(PDU *PDU_ptr, DbvResourceReportStatus eResourceReportStatus,
jmp_buf env)
```

```
int DBVset_snacc_resrepresp_report(PDU *PDU_ptr, DbvASN1External *psResourceReport, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_resrepresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_segment.c

Function for service user:

```
int DbvSegmentRequest(DbvAssocId lAssocId, DbvSegmentReq *psSegmentReq, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvSegReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_segrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psSegmentReq Checking Functions:

```
int DBVCheckDbvSegReqFields(DbvAssocId lAssocId, DbvSegmentReq *psSegmentReq, DbvError **ppsSIError)
DBVcheck_dbv_segreq_segrecs()
DBVcheck_dbv_segreq_otherinfo()

int DBVcheck_dbv_segreq_segrecs(DbvAssocId lAssocId, DbvSegmentReq *psSegmentReq, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_response_rec()

int DBVcheck_dbv_segreq_otherinfo(DbvAssocId lAssocId, DbvSegmentReq *psSegmentReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psSegmentReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_segrequest(PDU *PDU_ptr, DbvSegmentReq *psSegmentReq, jmp_buf env)
DBVset_snacc_segreq_refid()
DBVset_snacc_segreq_numrecs()
DBVset_snacc_segreq_segrecs()
DBVset_snacc_segreq_otherinfo()

int DBVset_snacc_segreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_segreq_numrecs(PDU *PDU_ptr, Ulong ulNumberOfRecordsReturned, jmp_buf env)

int DBVset_snacc_segreq_segrecs(PDU *PDU_ptr, DbvNamePlusRecord **ppsSegmentRecords, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_resprec()

int DBVset_snacc_segreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
```

```
DBVdbv_to_snacc_otherinfo()
```

origin_z3950_extend.c:

Function for service user:

```
int DbvExtendedServicesRequest(DbvAssocId lAssocId, DbvExtendedServicesReq *psExtendedServicesReq,
DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVCheckDbvExtendReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_extendrequest()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psExtendedServicesReq Checking Functions:

```
int DBVCheckDbvExtendReqFields(DbvAssocId lAssocId, DbvExtendedServicesReq *psExtendedServicesReq,
DbvError **ppsSIError)
DBVcheck_dbv_extendreq_funct()
DBVcheck_dbv_extendreq_packtype()
DBVcheck_dbv_extendreq_rettime()
DBVcheck_dbv_extendreq_permiss()
DBVcheck_dbv_extendreq_origspec()
DBVcheck_dbv_extendreq_waact()
DBVcheck_dbv_extendreq_otherinfo()

int DBVcheck_dbv_extendreq_funct(DbvAssocId lAssocId, DbvExtendedServicesReq *psExtendedServicesReq,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_extendreq_packtype(DbvAssocId lAssocId, DbvExtendedServicesReq
*psExtendedServicesReq, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_validoid()
DBVdbv_oid_error()

int DBVcheck_dbv_extendreq_rettime(DbvAssocId lAssocId, DbvExtendedServicesReq
*psExtendedServicesReq, DbvError **ppsSIError)
DBVcheck_dbv_intunit()

int DBVcheck_dbv_extendreq_permiss(DbvAssocId lAssocId, DbvExtendedServicesReq
*psExtendedServicesReq, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_allowfuncs()

int DBVcheck_dbv_extendreq_origspec(DbvAssocId lAssocId, DbvExtendedServicesReq
*psExtendedServicesReq, DbvError **ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_extendreq_waact(DbvAssocId lAssocId, DbvExtendedServicesReq *psExtendedServicesReq,
DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_extendreq_otherinfo(DbvAssocId lAssocId, DbvExtendedServicesReq
*psExtendedServicesReq, DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psExtendedServicesReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_extendrequest(PDU *PDU_ptr, DbvExtendedServicesReq *psExtendedServicesReq,
jmp_buf env)
DBVset_snacc_extendreq_refid()
DBVset_snacc_extendreq_function()
DBVset_snacc_extendreq_packtype()
DBVset_snacc_extendreq_packname()
DBVset_snacc_extendreq_userid()
DBVset_snacc_extendreq_retenttime()
DBVset_snacc_extendreq_permission()
DBVset_snacc_extendreq_description()
DBVset_snacc_extendreq_ospecoparam()
DBVset_snacc_extendreq_waitaction()
DBVset_snacc_extendreq_elements()
DBVset_snacc_extendreq_otherinfo()

int DBVset_snacc_extendreq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_extendreq_function(PDU *PDU_ptr, DbvExtServicesFunctions eFunction, jmp_buf env)

int DBVset_snacc_extendreq_packtype(PDU *PDU_ptr, DbvObjectIdentifier pcPackageType, jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_extendreq_packname(PDU *PDU_ptr, DbvInternationalString *psPackageName, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_extendreq_userid(PDU *PDU_ptr, DbvInternationalString *psUserId, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_extendreq_retenttime(PDU *PDU_ptr, DbvIntUnit *psRetentionTime, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_intunit()

int DBVset_snacc_extendreq_permission(PDU *PDU_ptr, DbvPermissions **ppsPermissions, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_permissions()

int DBVset_snacc_extendreq_description(PDU *PDU_ptr,  DbvInternationalString *psDescription, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_extendreq_ospecoparam(PDU *PDU_ptr, DbvASN1External *psOriginSpecificParameter,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_extendreq_waitaction(PDU *PDU_ptr, DbvExtServicesWaitAction eWaitAction, jmp_buf
env)

int DBVset_snacc_extendreq_elements(PDU *PDU_ptr, DbvInternationalString  *psElements, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_extendreq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

target_z3950_extend.c:

Function for service user:

```
int DbvExtendedServicesResponse(DbvAssocId lAssocId, DbvExtendedServicesRsp *psExtendedServicesRsp,
DbvError **ppsSIError)
DBVEnterFuncTrace()
```

```
DBVCheckDbvExtendRspFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_extendresponse()
DBVfill_error_structure()
DBVencode_and_send()
DBVLeaveFuncTrace()
```

psExtendedServicesRsp Checking Functions:

```
int DBVCheckDbvExtendRspFields(DbvAssocId lAssocId, DbvExtendedServicesRsp *psExtendedServicesRsp,
DbvError **ppsSIError)
DBVcheck_dbv_extendrsp_opstat()
DBVcheck_dbv_extendrsp_diags()
DBVcheck_dbv_extendrsp_packtype()
DBVcheck_dbv_extendrsp_otherinfo()

int DBVcheck_dbv_extendrsp_opstat(DbvAssocId lAssocId, DbvExtendedServicesRsp
*psExtendedServicesRsp, DbvError **ppsSIError)
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_extendrsp_diags(DbvAssocId lAssocId, DbvExtendedServicesRsp *psExtendedServicesRsp,
DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_multdiag()

int DBVcheck_dbv_extendrsp_packtype(DbvAssocId lAssocId, DbvExtendedServicesRsp
*psExtendedServicesRsp, DbvError **ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_extendrsp_otherinfo(DbvAssocId lAssocId, DbvExtendedServicesRsp
*psExtendedServicesRsp, DbvError **ppsSIError)
DBVcheck_dbv_otherinfo()
```

psExtendedServicesRsp to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_extendresponse(PDU *PDU_ptr, DbvExtendedServicesRsp *psExtendedServicesRsp,
jmp_buf env)
DBVset_snacc_extendresp_refid()
DBVset_snacc_extendresp_opstat()
DBVset_snacc_extendresp_diagnost()
DBVset_snacc_extendresp_taskpack()
DBVset_snacc_extendresp_otherinfo()

int DBVset_snacc_extendresp_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_extendresp_opstat(PDU *PDU_ptr, DbvExtServicesOperationStatus eOperationStatus,
jmp_buf env)

int DBVset_snacc_extendresp_diagnost(PDU *PDU_ptr, DbvDiagRec **ppsDiagnostics, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_multdiag()

int DBVset_snacc_extendresp_taskpack(PDU *PDU_ptr, DbvASN1External *psTaskPackage, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_extendresp_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

Close PDU:

origin z3950_close.c:

psCloseReq to snacc Structure Mapping Functions:

```
int DBVset_dbv_originfields_closereq(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_closereq_refid()
DBVset_dbv_closereq_closereason()
DBVset_dbv_closereq_diaginfo()
DBVset_dbv_closereq_resrepform()
DBVset_dbv_closereq_resreport()
DBVset_dbv_closereq_otherinfo()
FreePDU()
```

target z3950_close.c:

psCloseReq to snacc Structure Mapping Functions:

```
int DBVset_dbv_targetfields_closereq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_closereq_refid()
DBVset_dbv_closereq_closereason()
DBVset_dbv_closereq_diaginfo()
DBVset_dbv_closereq_resrepform()
DBVset_dbv_closereq_resreport()
DBVset_dbv_closereq_otherinfo()
FreePDU()
```

shared z3950_close.c:

Function for service user:

```
int DbvCloseRequest(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVClose()
DBVLeaveFuncTrace()

int DBVClose(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError **ppsSIError)
DBVCheckDbvCloseReqFields()
SIWhichStack()
SL_Current()
Asn1Alloc()
CheckAsn1Alloc()
DBVsetup_snacc_closerequest()
DBVfill_error_structure()
DBVencode_and_send()
```

psCloseReq Checking Functions:

```
int DBVCheckDbvCloseFields(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError **ppsSIError)
DBVcheck_dbv_close_reason()
DBVcheck_dbv_close_resrepfmt()
DBVcheck_dbv_close_resrep()
DBVcheck_dbv_close_otherinfo()

int DBVcheck_dbv_close_reason(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError **ppsSIError)
```

```
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_close_resrepfmt(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError
**ppsSIError)
DBVcheck_dbv_resrepoid()
DBVdbv_oid_error()

int DBVcheck_dbv_close_resrep(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError **ppsSIError)
DBVcheck_dbv_external()

int DBVcheck_dbv_close_otherinfo(DbvAssocId lAssocId, DbvCloseReq *psCloseReq, DbvError
**ppsSIError)
DBVcheck_dbv_otherinfo()
```

psCloseReq to snacc Structure Mapping Functions:

```
int DBVsetup_snacc_closerequest(PDU *PDU_ptr, DbvCloseReq *psCloseReq, jmp_buf env)
DBVset_snacc_closereq_refid()
DBVset_snacc_closereq_closereason()
DBVset_snacc_closereq_diaginfo()
DBVset_snacc_closereq_resrepform()
DBVset_snacc_closereq_resreport()
DBVset_snacc_closereq_otherinfo()

int DBVset_snacc_closereq_refid(PDU *PDU_ptr, DbvOctetString *psReferenceId, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVset_snacc_closereq_closereason(PDU *PDU_ptr, DbvCloseReason eCloseReason, jmp_buf env)

int DBVset_snacc_closereq_diaginfo(PDU *PDU_ptr, DbvInternationalString *psDiagnosticInformation,
jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVset_snacc_closereq_resrepform(PDU *PDU_ptr, DbvObjectIdentifier  pcResourceReportFormat,
jmp_buf env)
ASN_MakeOid()

int DBVset_snacc_closereq_resreport(PDU *PDU_ptr, DbvASN1External *psResourceReport, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()

int DBVset_snacc_closereq_otherinfo(PDU *PDU_ptr, DbvOtherInformation *psOtherInformation, jmp_buf
env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_otherinfo()
```

Function for service user:

```
int DbvCloseResponse(DbvAssocId lAssocId, DbvCloseReq *psCloseRsp, DbvError **ppsSIError)
DBVEnterFuncTrace()
DBVClose()
DBVLeaveFuncTrace()
```

DbvReceiveDataOrigin mapping of snacc structure to dbv structure

origin z3950 main.c

```
int DbvReceiveDataOrigin( DbvAssocId lAssocId, DbvSecs lSeconds, DbvOriginData **ppsOriginData,
DbvError **ppsError)
DBVEnterFuncTrace()
SIWhichStack()
SL_Current()
DBVreceive_data_over_socket()
DBVreceive_data_over_OSI()
DBVconvert_pdu_to_origin_data()
DBVfill_error_structure()
DBVLeaveFuncTrace()
```

si utilities.c:

```
void DBVEnterFuncTrace(char *function_name, DbvAssocId lAssocId)
DBVTraceLevel()
DBVTrace()
```

si utilities.c:

```
int SIWhichStack(DbvAssocId lAssocId, DbvProtocolStack *peProtocolStack, DbvError **ppsError)
DBVEnterFuncTrace()
DBVfill_error_structure()
SL_Locate()
DBVLeaveFuncTrace()
```

error.c:

```
void DBVfill_error_structure(  DbvAssocId lAssocId, DbvErrorClass eSIErrorClass, DbvAbortSource
eAbortSource, DbvAbortReason lAbortReason, char *pcDiagnostics, DbvErrorCode lIfOsCoError,
DbvAssociationState eAssociationState , DbvError **ppsSIError)
DbvAlloc()
DBVErrorTrace()
```

sl list.c:

```
void*   SL_Locate( SL List, SL_LocateFunc LocateFunc, void*Item )
SL_EmptyList()
SL_Start()
SL_Succ()
SL_Current()
SL_IsEnd ()

Bool_  SL_EmptyList(SL List)

void SL_Start(SL List)

void SL_Succ(SL List)
SError_()
```

memcheck.c:

```
void SError_( enum ErrorLevel_ Error, char*ErrorString, char*ModuleName )
```

si list.c:

```
void* SL_Current(SL List)

Bool_ SL_IsEnd(SL List)
```

si_utilities.c:

```
void DBVLeaveFuncTrace(char *function_name, DbvAssocId lAssocId, int return_value)
DBVTraceLevel()
DBVTrace()
```

z3950_send_receive.c:

```
int DBVreceive_data_over_socket(DbvAssocId lAssocId, PDU *PDU_ptr, int socket, DbvSecs lSeconds,
char **ber_data, unsigned long *ber_data_length, DbvError **ppsError)
DBVfill_error_structure()
IsCompleteBER()
DBVdecode_bytes_to_pdu()
```

berutil.c:

```
int IsCompleteBER(CHAR far *ptr, long len, long *remainder)
get_tag()
get_len()
IsCompleteBER()

int get_tag(int *tag, CHAR far *s, long len)

int get_len(long *fieldlen, CHAR far *s, long len)
```

z3950_enc_dec.c:

```
int DBVdecode_bytes_to_pdu(DbvAssocId lAssocId, char *encoded_pdu, int encoded_pdu_length, PDU
*pdu_ptr, DbvError **ppsSIError)
DBVinit_external_data()
SBufInstallData()
BDecPDU()
DBVfill_error_structure()


int DBVreceive_data_over_OSI(DbvAssocId lAssocId, PDU *PDU_ptr, int lOSIAssocId, DbvSecs lSeconds,
char **ber_data, unsigned long *ber_data_length, DbvError **ppsError)
DBVEnterFuncTrace()
DBVAllocCheck()
DBVAlloc()
DBVdecode_bytes_to_pdu()
DBVFree()
DBVLeaveFuncTrace()
```

origin_z3950_main.c

```
int DBVconvert_pdu_to_origin_data(DbvAssocId lAssocId, DbvOriginData *psOriginDa
ta, PDU *PDU_ptr, char *ber_data, unsigned long ber_data_length, jmp_buf env)
DBVTraceLevel()
DBVTrace()
DBVset_dbv_fields_initresp()
DBVset_dbv_fields_searchresp()
DBVset_dbv_fields_presresp()
DBVset_dbv_fields_deleteresultsetresp()
DBVset_dbv_fields_resrepresp()
DBVset_dbv_fields_scanresp()
DBVset_dbv_fields_sortresp()
DBVset_dbv_fields_segreq()
DBVset_dbv_fields_extendresp()
DBVset_dbv_fields_originfields_closereq()
DbvAlloc()
CheckDbvAlloc()
```

(InitRsp :origin z3950_init.c) :

```
int DBVset_dbv_fields_initresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
CheckDbvAlloc()
int DBVset_dbv_initresp_refid()
int DBVset_dbv_initresp_prover()
int DBVset_dbv_initresp_options()
int DBVset_dbv_initresp_prefmes()
int DBVset_dbv_initresp_maxmes()
int DBVset_dbv_initresp_result()
int DBVset_dbv_initresp_impid()
int DBVset_dbv_initresp_impname()
int DBVset_dbv_initresp_impver()
int DBVset_dbv_initresp_userinfo()
int DBVset_dbv_initresp_otherinfo()
FreePDU()

int DBVset_dbv_initresp_refid(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_initresp_prover(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
GetAsnBit()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_initresp_options(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
GetAsnBit()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_extendresp_refid(DbvExtendedServicesRsp *extendresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_extendresp_opstat(DbvExtendedServicesRsp *extendresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_extendresp_diagnost(DbvExtendedServicesRsp *extendresp, PDU *PDU_ptr, jmp_buf env)
AsnListCount()

int DBVset_dbv_initresp_prefmes(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_initresp_maxmes(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_initresp_result(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_initresp_impid(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initresp_impname(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initresp_impver(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initresp_userinfo(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_initresp_otherinfo(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(SearchRsp :origin z3950_search.c) :

```
int DBVset_dbv_fields_searchresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_searchresp_refid()
DBVset_dbv_searchresp_rescount()
DBVset_dbv_searchresp_numrecs()
DBVset_dbv_searchresp_nextres()
DBVset_dbv_searchresp_srchstat()
DBVset_dbv_searchresp_resstat()
DBVset_dbv_searchresp_presstat()
DBVset_dbv_searchresp_records()
DBVset_dbv_searchresp_addinfo()
DBVset_dbv_searchresp_otherinfo()
FreePDU()

int DBVset_dbv_searchresp_refid(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_searchresp_rescount(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchresp_numrecs(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchresp_nextres(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchresp_srchstat(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchresp_resstat(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_searchresp_presstat(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_searchresp_records(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_records()

int DBVset_dbv_searchresp_addinfo(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()

int DBVset_dbv_searchresp_otherinfo(DbvSearchRsp *searchresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(PresRsp:origin z3950_present.c) :

```
int DBVset_dbv_fields_presresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
int DBVset_dbv_presresp_refid()
int DBVset_dbv_presresp_numrecs()
int DBVset_dbv_presresp_nxtresset()
int DBVset_dbv_presresp_presstat()
int DBVset_dbv_presresp_records()
int DBVset_dbv_presresp_otherinfo()
FreePDU()


int DBVset_dbv_presresp_refid(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)
```

```
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_presresp_numrecs(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_presresp_nxtresset(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_presresp_presstat(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_presresp_records(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_records()

int DBVset_dbv_presresp_otherinfo(DbvPresentRsp *presresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(DelResultSetRsp: origin z3950_delete.c)

```
int DBVset_dbv_fields_deleteresultsetresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DBVset_dbv_delresp_refid()
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_delresp_opstat()
DBVsnacc_to_dbv_delsetstat()
DBVset_dbv_delresp_delliststat()
DBVsnacc_to_dbv_liststat()
DBVset_dbv_delresp_numnotdel()
DBVset_dbv_delresp_bulkliststat)
DBVset_dbv_delresp_delmess()
DBVset_dbv_delresp_otherinfo()
FreePDU()

int DBVset_dbv_delresp_refid(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

DBVset_dbv_delresp_opstat(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
DBVsnacc_to_dbv_delsetstat()

int DBVsnacc_to_dbv_delsetstat(DbvDeleteSetStatus *dbv_stat, DeleteSetStatus *snacc_stat, jmp_buf
env)

DBVset_dbv_delresp_delliststat(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_liststat()

int DBVsnacc_to_dbv_liststat(DbvListStatus *psDeleteListStatuses, ListStatusesSeq *data_ptr, jmp_buf
env)
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_delsetstat()

DBVset_dbv_delresp_numnotdel(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

DBVset_dbv_delresp_bulkliststat(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_liststat()

DBVset_dbv_delresp_delmess(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
```

```
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

DBVset_dbv_delresp_otherinfo(DbvDeleteResultSetRsp *delresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(SortRsp: origin z3950_sort.c)

```
int DBVset_dbv_fields_sortresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
int DBVset_dbv_sortresp_refid(DbvSortRsp *sortresp, PDU *PDU_ptr, jmp_buf env)
int DBVset_dbv_sortresp_sortstat(DbvSortRsp *sortresp, PDU *PDU_ptr, jmp_buf env)
int DBVset_dbv_sortresp_resstat(DbvSortRsp *sortresp, PDU *PDU_ptr, jmp_buf env)
int DBVset_dbv_sortresp_diag(DbvSortRsp *sortresp, PDU *PDU_ptr, jmp_buf env)
int DBVset_dbv_sortresp_otherinfo(DbvSortRsp *sortresp, PDU *PDU_ptr, jmp_buf env)
```

(ScanRsp: origin z3950_scan.c)

```
int DBVset_dbv_fields_scanresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_scanresp_refid()
DBVset_dbv_scanresp_stepsize()
DBVset_dbv_scanresp_scanstat()
DBVset_dbv_scanresp_numentries()
DBVset_dbv_scanresp_posterm()
DBVset_dbv_scanresp_entries()
DBVsnacc_to_dbv_entries()
DBVsnacc_to_dbv_entry()
DBVsnacc_to_dbv_terminfo()
DBVsnacc_to_dbv_attrtermlist()
DBVsnacc_to_dbv_occurbyatt()
DBVsnacc_to_dbv_occurbyattseq()
DBVsnacc_to_dbv_occurrence()
DBVsnacc_to_dbv_bydatabase()
DBVsnacc_to_dbv_bigoccurrthing()
DBVset_dbv_scanresp_attsetid()
DBVset_dbv_scanresp_otherinfo()
FreePDU()

int DBVset_dbv_scanresp_refid(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_scanresp_stepsize(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanresp_scanstat(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_scanresp_numentries(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_scanresp_posterm(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanresp_entries(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
AsnListCount()
DBVsnacc_to_dbv_entries()
DBVsnacc_to_dbv_multdiag()

int DBVsnacc_to_dbv_entries(DbvEntry **ppsEntry, ListEntriesSeqOf *entries, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
```

```
CheckDbvAlloc()
DBVsnacc_to_dbv_entry()

int DBVsnacc_to_dbv_entry(DbvEntry *psEntry, Entry *data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_terminfo()
DBVsnacc_to_dbv_diag_rec()

int DBVsnacc_to_dbv_terminfo(DbvTermInfo *psTermInfo, struct TermInfo* termInfo, jmp_buf env)
DBVsnacc_to_dbv_term()
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attribute_list()
DBVsnacc_to_dbv_attrtermlist()
DBVsnacc_to_dbv_occurbyatt()
DBVsnacc_to_dbv_otherinfo()

int DBVsnacc_to_dbv_attrtermlist(DbvAttributesPlusTerm **ppsAlternativeTerm, TermInfoSeqOf
*alternativeTerm, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attrterm()

int DBVsnacc_to_dbv_occurbyatt(DbvOccurrenceByAttributes *psByAttributes, OccurrenceByAttributes
*byAttributes, jmp_buf env)
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_occurbyattseq()

int DBVsnacc_to_dbv_occurbyattseq(DbvOccurrenceByAttributes *psByAttributes,
OccurrenceByAttributesSeq *data_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attribute_list()
DBVsnacc_to_dbv_occurrence()
DBVsnacc_to_dbv_otherinfo()

int DBVsnacc_to_dbv_occurrence(DbvOccurrence *psOccurrence, struct OccurrenceByAttributesSeqChoice*
occurrences, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_bydatabase()

int DBVsnacc_to_dbv_bydatabase(DbvOccurrenceByDb **ppsOccurrenceByDb,
OccurrenceByAttributesSeqChoiceSeqOf *byDatabase, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_bigoccurrthing()

int DBVsnacc_to_dbv_bigoccurrthing(DbvOccurrenceByDb *psOccurrenceByDb,
OccurrenceByAttributesSeqChoiceSeqOfSeq *data_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()

int DBVset_dbv_scanresp_attsetid(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanresp_otherinfo(DbvScanRsp *scanresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(ResRepRsp: origin z3950_resrep.c)

```
int DBVset_dbv_fields_resrepresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckAsn1Alloc()
DBVset_dbv_resrepresp_refid()
DBVset_dbv_resrepresp_repstat()
DBVset_dbv_resrepresp_report()
DBVset_dbv_resrepresp_otherinfo()
FreePDU()

int DBVset_dbv_resrepresp_refid(DbvResourceReportRsp *resrepresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_resrepresp_repstat(DbvResourceReportRsp *resrepresp, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_resrepresp_report(DbvResourceReportRsp *resrepresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_resrepresp_otherinfo(DbvResourceReportRsp *resrepresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(segmentReq: origin z3950_segment.c)

```
int DBVset_dbv_fields_segreq(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_segreq_refid()
DBVset_dbv_segreq_numrecs()
DBVset_dbv_segreq_segrecs()
DBVset_dbv_segreq_otherinfo()
FreePDU()

int DBVset_dbv_segreq_refid(DbvSegmentReq *segreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_segreq_numrecs(DbvSegmentReq *segreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_segreq_segrecs(DbvSegmentReq *segreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resprec()

int DBVset_dbv_segreq_otherinfo(DbvSegmentReq *segreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(ExtendedServicesRsp: origin z3950_extend.c)

```
int DBVset_dbv_fields_extendresp(DbvOriginData *psOriginData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_extendresp_refid()
DBVset_dbv_extendresp_opstat()
DBVset_dbv_extendresp_diagnost()
DBVset_dbv_extendresp_taskpack()
DBVset_dbv_extendresp_otherinfo()
FreePDU()
```

```
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_multdiag()

int DBVset_dbv_extendresp_taskpack(DbvExtendedServicesRsp *extendresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_extendresp_otherinfo(DbvExtendedServicesRsp *extendresp, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

DbvReceiveDataTarget mapping of snacc structure to dbv structure

target z3950 main.c

```
int DbvReceiveDataTarget( DbvAssocId lAssocId, DbvSecs lSeconds, DbvTargetData **ppsTargetData,
DbvError **ppsError)
DBVEnterFuncTrace()
SIWhichStack()
SL_Current()
DBVreceive_data_over_socket()
DBVreceive_data_over_OSI()
DBVconvert_pdu_to_target_data()
DBVfill_error_structure()
DBVAllocCheck()
DBVAlloc()
DBVLeaveFuncTrace()
```

(InitReq :target z3950 init.c) :

```
int DBVset_dbv_fields_initreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_initreq_refid()
DBVset_dbv_initreq_prover()
DBVset_dbv_initreq_options()
DBVset_dbv_initreq_prefmes()
DBVset_dbv_initreq_maxmes()
DBVset_dbv_initreq_authentication()
DBVset_dbv_initreq_authentication_open()
DBVset_dbv_initreq_authentication_pass()
DBVset_dbv_initreq_authentication_other()
DBVset_dbv_initreq_impid()
DBVset_dbv_initreq_impname()
DBVset_dbv_initreq_impver()
DBVset_dbv_initreq_userinfo()
DBVset_dbv_initreq_otherinfo()
FreePDU()

int DBVset_dbv_initreq_refid(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_initreq_prover(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
GetAsnBit()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_initreq_options(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
GetAsnBit()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_initreq_prefmes(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_initreq_maxmes(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_initreq_authentication(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_initreq_authentication_open()
DBVset_dbv_initreq_authentication_pass()
DBVset_dbv_initreq_authentication_other()

int DBVset_dbv_initreq_authentication_open(PDU *PDU_ptr, DbvInitializeReq *initreq, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initreq_authentication_pass(PDU *PDU_ptr, DbvInitializeReq *initreq, jmp_buf env)
DbvAlloc()
```

```
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initreq_authentication_other(PDU *PDU_ptr, DbvInitializeReq *initreq, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_initreq_impid(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initreq_impname(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initreq_impver(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_initreq_userinfo(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_initreq_otherinfo(DbvInitializeReq *initreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(SearchReq:target z3950 search.c) :

```
int DBVset_dbv_fields_searchreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_searchreq_refid()
DBVset_dbv_searchreq_ssetupper()
DBVset_dbv_searchreq_lsetlower()
DBVset_dbv_searchreq_msetpres()
DBVset_dbv_searchreq_replace()
DBVset_dbv_searchreq_resultset()
DBVset_dbv_searchreq_dbnames()
DBVset_dbv_searchreq_ssetelname()
DBVset_dbv_searchreq_msetelname()
DBVset_dbv_searchreq_prefrecsyn()
DBVset_dbv_searchreq_query()
DBVsnacc_to_dbv_query()
DBVsnacc_to_dbv_rpn()
DBVsnacc_to_dbv_operand()
DBVsnacc_to_dbv_resulattr()
DBVsnacc_to_dbv_lopropop()
DBVsnacc_to_dbv_operator()
DBVsnacc_to_dbv_prox()
DBvsnacc_to_dbv_proxunitcode()
DBVset_dbv_searchreq_addinfo()
DBVset_dbv_searchreq_otherinfo()
FreePDU()

int DBVset_dbv_searchreq_refid(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_searchreq_ssetupper(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchreq_lsetlower(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchreq_msetpres(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
```

```
int DBVset_dbv_searchreq_replace(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_searchreq_resultset(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_searchreq_dbnames(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()


int DBVset_dbv_searchreq_ssetelname(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_elset_name()

int DBVset_dbv_searchreq_msetelname(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_elset_name()

int DBVset_dbv_searchreq_prefrecsyn(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
```

oid tool.c:

```
char *ASN_EncodedToString(AsnOid *v)
```

target z3950 search.c:

```
int DBVset_dbv_searchreq_query(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()
DBVsnacc_to_dbv_query()

int DBVsnacc_to_dbv_query(DbvRPNQuery *psRpnQuery, struct RPNQuery* type_1, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_rpn()

int DBVsnacc_to_dbv_rpn(DbvRPNStructure *psRpnStructure, struct RPNStructure* rpn, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_operand()
DBVsnacc_to_dbv_lopropop()

int DBVsnacc_to_dbv_operand(DbvOperand *psRpnOperand, struct Operand* op, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attrterm()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_resulattr()

int DBVsnacc_to_dbv_resulattr(DbvResultSetPlusAttributes *psResuLtSetPlusAttributes, struct
ResultSetPlusAttributes* resultAttr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attribute_list()

int DBVsnacc_to_dbv_lopropop(DbvRPNLopRopOperator *psRpnLopRopOperator, struct RPNStructureSeq*
rpnRpnOp, jmp_buf env)
```

```
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_rpn()
DBVsnacc_to_dbv_operator()

int DBVsnacc_to_dbv_operator(DbvOperator *psOperator, struct Operator* op, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_prox()

int DBVsnacc_to_dbv_prox(DbvProximityOperator *psProximityOperator, struct ProximityOperator* prox,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_proxunitcode()

int DBvsnacc_to_dbv_proxunitcode(DbvProximityUnitCode *psProximityUnitCode, struct
ProximityOperatorChoice* proximityUnitCode, jmp_buf env)

int DBVset_dbv_searchreq_addinfo(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()

int DBVset_dbv_searchreq_otherinfo(DbvSearchReq *searchreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(PresReq: target z3950 present.c)

```
int DBVset_dbv_fields_presreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_presreq_refid()
DBVset_dbv_presreq_ressetid()
DBVset_dbv_presreq_ressetstart()
DBVset_dbv_presreq_numrecsreq()
DBVset_dbv_presreq_addranges()
DBVset_dbv_presreq_reccomp()
DBVsnacc_to_dbv_elset_name()
DBVsnacc_to_dbv_el_generic()
DBVsnacc_to_dbv_el_specific()
DBVsnacc_to_dbv_presreq_complex()
DBVsnacc_to_dbv_pres_dbspecific()
DBVsnacc_to_dbv_pres_recsyntax()
DBVset_dbv_presreq_prefrecsyn()
DBVset_dbv_presreq_maxsegcnt()
DBVset_dbv_presreq_maxrecsze()
DBVset_dbv_presreq_maxsegsze()
DBVset_dbv_presreq_otherinfo()
FreePDU()

int DBVset_dbv_presreq_refid(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_presreq_ressetid(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_presreq_ressetstart(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_presreq_numrecsreq(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_presreq_addranges(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
```

```
int DBVset_dbv_presreq_reccomp(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_elset_name()
DBVsnacc_to_dbv_presreq_complex()

int DBVsnacc_to_dbv_elset_name(DbvElementSetNames *psElementSetNames, struct ElementSetNames*
simple, jmp_buf env)
DBVsnacc_to_dbv_el_generic()
DBVsnacc_to_dbv_el_specific()

int DBVsnacc_to_dbv_el_generic(DbvElementSetNames *psElementSetNames, struct ElementSetNames*
simple, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_el_specific(DbvElementSetNames *psElementSetNames, struct ElementSetNames*
simple, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_presreq_complex(DbvCompSpec *psCompSpec, struct CompSpec* complexy, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_specification()
AsnListCount()
DBVsnacc_to_dbv_pres_dbspecific()
DBVsnacc_to_dbv_pres_recsyntax()

int DBVsnacc_to_dbv_pres_dbspecific(DbvDbSpecificComp **ppsDbSpecific, CompSpecSeqOf *dbSpecific,
jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_specification()

int DBVsnacc_to_dbv_pres_recsyntax(DbvObjectIdentifier *ppcRecordSyntax, CompSpecSeqOf1*
recordSyntax, jmp_buf env)
FOR_EACH_LIST_ELMT()
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_presreq_prefrecsyn(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_presreq_maxsegcnt(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_presreq_maxrecsze(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_presreq_maxsegsze(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_presreq_otherinfo(DbvPresentReq *presreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(DelResultSetReq: target_z3950_delete.c)

```
int DBVset_dbv_fields_delreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_delreq_refid()
DBVset_dbv_delreq_deletefunc()
DBVset_dbv_delreq_ressetid()
DBVset_dbv_delreq_otherinfo()
FreePDU()

int DBVset_dbv_delreq_refid(DbvDeleteResultSetReq *delreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_delreq_deletefunc(DbvDeleteResultSetReq *delreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_delreq_ressetid(DbvDeleteResultSetReq *delreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_delreq_otherinfo(DbvDeleteResultSetReq *delreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(SortReq: target_z3950_sort.c)

```
int DBVset_dbv_fields_sortreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_sortreq_refid()
DBVset_dbv_sortreq_inresset()
DBVset_dbv_sortreq_sortedres()
DBVset_dbv_sortreq_sortseq()
DBVsnacc_to_dbv_sortkeyspec()
DBVsnacc_to_dbv_sortelmnt()
DBVsnacc_to_dbv_sortkey()
DBVsnacc_to_dbv_soratts()
DBVsnacc_to_dbv_dbspecific()
DBVsnacc_to_dbv_specsortkey()
DBVsnacc_to_dbv_sortrel()
DBVsnacc_to_dbv_sortcase()
DBVsnacc_to_dbv_sortmv()
DBVset_dbv_sortreq_otherinfo()
FreePDU()

int DBVset_dbv_sortreq_refid(DbvSortReq *sortreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_sortreq_inresset(DbvSortReq *sortreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_sortreq_sortedres(DbvSortReq *sortreq, PDU *PDU_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_sortreq_sortseq(DbvSortReq *sortreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
```

```
DBVsnacc_to_dbv_sortkeyspec()

int DBVsnacc_to_dbv_sortkeyspec(DbvSortKeySpec *psSortSequence, SortKeySpec *data_ptr, jmp_buf env)
DBVsnacc_to_dbv_sortelmnt()
DBVsnacc_to_dbv_sortrel()
DBVsnacc_to_dbv_sortcase()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_sortmv()

int DBVsnacc_to_dbv_sortelmnt(DbvSortElement *psSortElement, struct SortElement* sortElement,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_sortkey()
AsnListCount()
DBVsnacc_to_dbv_dbspecific()

int DBVsnacc_to_dbv_sortkey(DbvSortKey *psGeneric, struct SortKey* generic, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_specification()
DBVsnacc_to_dbv_soratts()

int DBVsnacc_to_dbv_soratts(DbvSortAttributes *psSortAttributes, struct SortKeySeq* sortAttributes,
jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
AsnListCount()
DBVsnacc_to_dbv_attribute_list()

int DBVsnacc_to_dbv_dbspecific(DbvDbSpecificSortKey **ppsDatabaseSpecific, SortElementSeqOf
*datbaseSpecific, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_specsortkey()

int DBVsnacc_to_dbv_specsortkey(DbvDbSpecificSortKey *psDatabaseSpecific, SortElementSeqOfSeq
*data_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_sortkey()

int DBVsnacc_to_dbv_sortrel(DbvSortKeySpec *psSortSequence, SortKeySpec *data_ptr, jmp_buf env)

int DBVsnacc_to_dbv_sortcase(DbvSortKeySpec *psSortSequence, SortKeySpec *data_ptr, jmp_buf env)

int DBVsnacc_to_dbv_sortmv(DbvSortMVAction *psMVActionType, struct SortKeySpecChoice*
missingValueAction, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_sortreq_otherinfo(DbvSortReq *sortreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(ScanReq: target_z3950_scan.c)

```
int DBVset_dbv_fields_scanreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_scanreq_refid()
DBVset_dbv_scanreq_dbnames()
DBVset_dbv_scanreq_attsetid()
DBVset_dbv_scanreq_attplusterm()
```

```
DBVset_dbv_scanreq_stepsize()
DBVset_dbv_scanreq_numterms()
DBVset_dbv_scanreq_prefpos()
DBVset_dbv_scanreq_otherinfo()
FreePDU()

int DBVset_dbv_scanreq_refid(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_scanreq_dbnames(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_scanreq_attsetid(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanreq_attplusterm(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
DBVsnacc_to_dbv_attrterm()

int DBVset_dbv_scanreq_stepsize(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanreq_numterms(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_scanreq_prefpos(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_scanreq_otherinfo(DbvScanReq *scanreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

(ResRepReq: target_z3950_resrep.c)

```
int DBVset_dbv_fields_resrepreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_resrepreq_refid()
DBVset_dbv_resrepreq_opid()
DBVset_dbv_resrepreq_prefformat()
DBVset_dbv_resrepreq_otherinfo()
FreePDU()

int DBVset_dbv_resrepreq_refid(DbvResourceReportReq *resrepreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_resrepreq_opid(DbvResourceReportReq *resrepreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_resrepreq_prefformat(DbvResourceReportReq *resrepreq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_resrepreq_otherinfo(DbvResourceReportReq *resrepreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
```

```
DBVsnacc_to_dbv_otherinfo()
```

(ExtendedServiceReq: target_z3950_extend.c)

```
int DBVset_dbv_fields_extendreq(DbvTargetData *psTargetData, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVset_dbv_extendreq_refid()
DBVset_dbv_extendreq_function()
DBVset_dbv_extendreq_packtype()
DBVset_dbv_extendreq_packname()
DBVset_dbv_extendreq_userid()
DBVset_dbv_extendreq_retenttime()
DBVset_dbv_extendreq_permission()
DBVset_dbv_extendreq_description()
DBVset_dbv_extendreq_ospecoparam()
DBVset_dbv_extendreq_waitaction()
DBVset_dbv_extendreq_elements()
DBVset_dbv_extendreq_otherinfo()
FreePDU()

int DBVset_dbv_extendreq_refid(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_extendreq_function(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_extendreq_packtype(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_extendreq_packname(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_extendreq_userid(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_extendreq_retenttime(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_intunit()

int DBVset_dbv_extendreq_permission(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_permissions()

int DBVset_dbv_extendreq_description(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_extendreq_ospecoparam(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_extendreq_waitaction(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_extendreq_elements(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
```

```
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_extendreq_otherinfo(DbvExtendedServicesReq *extendreq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

DbvReceiveDataOrigin/DbvReceiveDataTarget shared mapping of snacc structure to dbv structure

DbvReceiveDataOrigin/DbvReceiveDataTarget shared mapping of snacc structure to dbv structure

(CloseReq: shared z3950_close.c)

```
int DBVset_dbv_closereq_refid(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVset_dbv_closereq_closereason(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)

int DBVset_dbv_closereq_diaginfo(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVset_dbv_closereq_resrepform(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()

int DBVset_dbv_closereq_resreport(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVset_dbv_closereq_otherinfo(DbvCloseReq *closereq, PDU *PDU_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_otherinfo()
```

z3950_otherinfo.c:

Common routines used when mapping Dbv otherinformation fields onto snacc otherinformation fields z3950_otherinfo.c: (used when sending Requests or Responses).

```
int DBVdbv_to_snacc_otherinfo(AsnList *snacc_otherinfo, DbvOtherInformation *psOtherInformation,
jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
ASN_MakeOid()
DBVdbv_to_snacc_otherinfo_char()
DBVdbv_to_snacc_otherinfo_bin()
DBVdbv_to_snacc_otherinfo_ext()
DBVdbv_to_snacc_otherinfo_oid()

int DBVdbv_to_snacc_otherinfo_char(struct OtherInformationSeq *data_ptr, DbvInternationalString
*psCharacterInfo, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_otherinfo_bin(struct OtherInformationSeq *data_ptr, DbvOctetString
*psBinaryInfo, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_otherinfo_ext(struct OtherInformationSeq *data_ptr, DbvASN1External
*psExternallyDefinedInfo, jmp_buf env)
Asn1Alloc
CheckAsn1Alloc()
```

```
DBVdbv_to_snacc_external()

int DBVdbv_to_snacc_otherinfo_oid(struct OtherInformationSeq *data_ptr, DbvObjectIdentifier
pcObjectIdentifier, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
ASN_MakeOid()
```

Common routines used when mapping snacc otherinformation fields onto Dbvotherinformation fields z3950_otherinfo.c: (used when receiving Requests or Responses).

```
int DBVsnacc_to_dbv_otherinfo(DbvOtherInformation *psOtherInformation, AsnList *snacc_otherinfo,
jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
ASN_EncodedToString()
DBVsnacc_to_dbv_otherinfo_char()
DBVsnacc_to_dbv_otherinfo_bin()
DBVsnacc_to_dbv_otherinfo_ext()
DBVsnacc_to_dbv_otherinfo_oid()

int DBVsnacc_to_dbv_otherinfo_char(DbvOtherInformation *psOtherInformation, OtherInformationSeq
*data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVsnacc_to_dbv_otherinfo_bin(DbvOtherInformation *psOtherInformation, OtherInformationSeq
*data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVsnacc_to_dbv_otherinfo_ext(DbvOtherInformation *psOtherInformation, OtherInformationSeq
*data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()

int DBVsnacc_to_dbv_otherinfo_oid(DbvOtherInformation *psOtherInformation, OtherInformationSeq
*data_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
```

Checking routines:

```
int DBVcheck_dbv_otherinfo(DbvAssocId lAssocId, DbvOtherInformation *psOtherInformation, char
*error_buffer, DbvError **ppsSIError)
DBVcheck_dbv_validoid()
DBVdbv_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVdbv_invalid_enum_error()
```

z3950_externals.c

Initialising externals:

Done once before encoding/decoding of first external (snacc structure->BER, BER->snacc structure):

```
int DBVinit_external_data()
```

```
InitAnyZ39_50_EXTERNALS_A_1994();
InitAnyZ39_50_EXTERNALS_B_1994();
InitAnyZ39_50_EXTERNALS_D_1994();
InitAnyZ39_50_EXTERNALS_E_1994();
InitAnyZ39_50_EXTERNALS_F_1994();
InitAnyZ39_50_EXTERNALS_ISO_1994();
```

Done after each BER send:

```
int DBV_reset_external_counters(void)
```

Special function for checking if an external is unknown (returns 1 if oid is of an unknown external, 0 if oid is of a known external)

```
int DBVunknown_external(char *oid)
```

Dbv to snacc mapping:

```
int DBVdbv_to_snacc_external(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
ASN_MakeOid()
DBVdbv_to_snacc_sutrs()
DBVdbv_to_snacc_isosutrs()
DBVdbv_to_snacc_opac()
DBVdbv_to_snacc_isoopac()
DBVdbv_to_snacc_summary()
DBVdbv_to_snacc_isosummary()
DBVdbv_to_snacc_resrep1()
DBVdbv_to_snacc_isoresrep1()
DBVdbv_to_snacc_resrep2()
DBVdbv_to_snacc_isoresrep2()
DBVdbv_to_snacc_estaskpack()
DBVdbv_to_snacc_isoestaskpack()
DBVdbv_to_snacc_update()
DBVdbv_to_snacc_isoupdate()
DBVdbv_to_snacc_aligned()
DBVdbv_to_snacc_arbitrary()

int DBVsnacc_to_dbv_external(DbvASN1External *dbv_external, NEWEXTERNAL *snacc_external, jmp_buf
env)
ASN_EncodedToStrin()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_singleasn1()
DBVsnacc_to_dbv_aligned()
DBVsnacc_to_dbv_arbitrary()

int DBVsnacc_to_dbv_singleasn1(char *direct_reference, DbvASN1External *dbv_external, NEWEXTERNAL
*snacc_external, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_sutrs()
DBVsnacc_to_dbv_isosutrs()
DBVsnacc_to_dbv_opac()
DBVsnacc_to_dbv_isoopac()
DBVsnacc_to_dbv_summary()
DBVsnacc_to_dbv_isosummary()
DBVsnacc_to_dbv_resrep1()
DBVsnacc_to_dbv_isoresrep1()
DBVsnacc_to_dbv_resrep2()
DBVsnacc_to_dbv_isoresrep2()
DBVsnacc_to_dbv_estaskpack()
DBVsnacc_to_dbv_isoestaskpack()
DBVsnacc_to_dbv_update()
DBVsnacc_to_dbv_isoupdate()
DBVsnacc_to_dbv_unknown()

int DBVdbv_to_snacc_sutrs(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
```

```
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_isosutrs(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
DBVdbv_to_snacc_inter_string()

int DBVsnacc_to_dbv_sutrs(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_isosutrs(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVdbv_to_snacc_opac(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_opac_main()

int DBVdbv_to_snacc_isoopac(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_opac_main()

int DBVdbv_to_snacc_opac_main(OrigOPACRecord *opac_ptr, struct DbvOpacRecord *psOpacRecord, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()
AsnListNew()
DBVdbv_to_snacc_holdings()

int DBVdbv_to_snacc_holdings(OrigOPACRecordSeqOf *holdingsData, DbvHoldingsRecord **ppsHoldingsData,
jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_holding_rec()

int DBVdbv_to_snacc_holding_rec(HoldingsRecord *data_ptr, DbvHoldingsRecord *psHoldingsData, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()
DBVdbv_to_snacc_holdings_bulk()
DBVdbv_to_snacc_hold_volume()
DBVdbv_to_snacc_hold_circ()

int DBVdbv_to_snacc_holdings_bulk(struct HoldingsAndCircData *holdingsAndCirc,
DbvHoldingsAndCircData sHoldingsAndCirc, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_hold_volume(struct HoldingsAndCircData *holdingsAndCirc, DbvHoldingsAndCircData
sHoldingsAndCirc, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_vol_struct()

int DBVdbv_to_snacc_vol_struct(Volume *data_ptr, DbvVolume *psVolume, jmp_buf env)
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_hold_circ(struct HoldingsAndCircData *holdingsAndCirc, DbvHoldingsAndCircData
sHoldingsAndCirc, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_circ_struct()
```

```
int DBVdbv_to_snacc_circ_struct(CircRecord *data_ptr, DbvCircRecord *psCircRecord, jmp_buf env)
DBVdbv_to_snacc_inter_string()


void DBVsnacc_to_dbv_opac(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_opac_main()

int DBVsnacc_to_dbv_isoopac(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_opac_main()

int DBVsnacc_to_dbv_opac_main(struct DbvOpacRecord *psOpacRecord, OrigOPACRecord *opac_ptr, jmp_buf
env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_holding()

int DBVsnacc_to_dbv_holding(DbvHoldingsRecord **ppsHoldingsData, OrigOPACRecordSeqOf *holdingsData,
jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()
DBVsnacc_to_dbv_holding_bulk()
DBVsnacc_to_dbv_hold_volume()
DBVsnacc_to_dbv_hold_circ()

int DBVsnacc_to_dbv_holding_bulk(DbvHoldingsAndCircData *psHoldingsAndCirc, struct
HoldingsAndCircData* holdingsAndCirc, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_hold_volume(DbvHoldingsAndCircData *psHoldingsAndCirc, struct
HoldingsAndCircData* holdingsAndCirc, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_hold_circ(DbvHoldingsAndCircData *psHoldingsAndCirc, struct HoldingsAndCircData*
holdingsAndCirc, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_circrec()

int DBVsnacc_to_dbv_circrec(DbvCircRecord *psCircRecord, CircRecord *data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVdbv_to_snacc_summary(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_summ_main()

int DBVdbv_to_snacc_isosummary(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_summ_main()
```

```
int DBVdbv_to_snacc_summ_main(BriefBib *summary_ptr, struct DbvSummaryRecord *psSummaryRecord,
jmp_buf env)
DBVdbv_to_snacc_inter_string()
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_summ_format()
Asn1Alloc()
DBVdbv_to_snacc_otherinfo()

int DBVdbv_to_snacc_summ_format(BriefBibSeqOf *format, DbvFormatSpec **ppsFormat, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVsnacc_to_dbv_summary(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_summ_main()

int DBVsnacc_to_dbv_isosummary(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy
single_ASN1_type, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_summ_main()

int DBVsnacc_to_dbv_summ_main(struct DbvSummaryRecord *psSummaryRecord, BriefBib *summary_ptr,
jmp_buf env)
DBVsnacc_to_dbv_inter_string()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_summ_format()
DBVsnacc_to_dbv_otherinfo()

int DBVsnacc_to_dbv_summ_format(DbvFormatSpec **ppsFormat, BriefBibSeqOf *format, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVdbv_to_snacc_resrep1(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
DBVdbv_to_snacc_resrep1_main()

int DBVdbv_to_snacc_isoresrep1(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
DBVdbv_to_snacc_resrep1_main()

int DBVdbv_to_snacc_resrep1_main(ResourceReport1 *resrep1_ptr, struct DbvResourceReport1
*psResourceReport1, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_estimate1()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_estimate1(Estimate *data_ptr, DbvEstimate1 *psEstimate, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_resrep1(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
DBVdbv_to_snacc_resrep1_main()

int DBVdbv_to_snacc_isoresrep1(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
DBVdbv_to_snacc_resrep1_main()

int DBVdbv_to_snacc_resrep1_main(ResourceReport1 *resrep1_ptr, struct DbvResourceReport1
*psResourceReport1, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
```

```
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_estimate1()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_estimate1(Estimate *data_ptr, DbvEstimate1 *psEstimate, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVsnacc_to_dbv_resrep1(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resrep1_main()

int DBVsnacc_to_dbv_isoresrep1(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy
single_ASN1_type, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resrep1_main()

int DBVsnacc_to_dbv_resrep1_main(struct DbvResourceReport1 *psResourceReport1, ResourceReport1
*resrep1_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_estimate1()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_estimate1(DbvEstimate1 *psEstimate, Estimate *data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVdbv_to_snacc_resrep2(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
Asn1Alloc
CheckAsn1Alloc()
DBVdbv_to_snacc_resrep2_main()

int DBVdbv_to_snacc_isoresrep2(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_resrep2_main()

int DBVdbv_to_snacc_resrep2_main(ResourceReport2 *resrep2_ptr, struct DbvResourceReport2
*psResourceReport2, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_resrep2est()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_resrep2est(ResourceReport2SeqOf *estimates, DbvEstimate2 **ppsEstimate, jmp_buf
env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_estimate2()

int DBVdbv_to_snacc_estimate2(Estimate2 *data_ptr, DbvEstimate2 *psEstimate, jmp_buf env)
Asn1Alloc()
CheckAsnheckAsn1Alloc(
DBVdbv_to_snacc_strornum()
DBVdbv_to_snacc_intunit()

int DBVsnacc_to_dbv_resrep2(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resrep2_main()

int DBVsnacc_to_dbv_isoresrep2(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy
single_ASN1_type, jmp_buf env)
DbvAlloc()
```

```
CheckDbvAlloc()
DBVsnacc_to_dbv_resrep2_main()

int DBVsnacc_to_dbv_resrep2_main(struct DbvResourceReport2 *psResourceReport2, ResourceReport2
*resrep2_ptr, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resrep2est()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_resrep2est(DbvEstimate2 **ppsEstimate, ResourceReport2SeqOf *estimates, jmp_buf
env)
FOR_EACH_LIST_ELMT()
CheckDbvAlloc()
DBVsnacc_to_dbv_estimate2()

int DBVsnacc_to_dbv_estimate2(DbvEstimate2 *psEstimate, Estimate2 *data_ptr, jmp_buf env)
DBVsnacc_to_dbv_strornum()
DBVsnacc_to_dbv_intunit()

int DBVdbv_to_snacc_estaskpack(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_taskpack_main()

int DBVdbv_to_snacc_isoestaskpack(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_taskpack_main()

int DBVdbv_to_snacc_taskpack_main(TaskPackage *taskpack_ptr, struct DbvEsTaskPackage
*psEsTaskPackage, jmp_buf env)
ASN_MakeOid()
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_intunit()
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_permissions()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_octet_string()
AsnListNew()
DBVdbv_to_snacc_diagrec_list()
DBVdbv_to_snacc_external()

int DBVdbv_to_snacc_permissions(Permissions *permissions, DbvPermissions **ppsPermissions, jmp_buf
env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_permisnode()

int DBVdbv_to_snacc_permisnode(PermissionsSeq *data_ptr, DbvPermissions *psPermissions, jmp_buf env)
DBVdbv_to_snacc_inter_string()
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_allowfuncts()

int DBVdbv_to_snacc_allowfuncts(PermissionsSeqSeqOf *allowableFunctions, DbvPermAllowableFunctions
**eAllowableFunctions, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_diagrec_list(AsnList *snacc_diaglist, DbvDiagRec **dbv_diaglist, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_diag_rec()
```

```
int DBVsnacc_to_dbv_estaskpack(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy
single_ASN1_type, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_taskpack_main()

int DBVsnacc_to_dbv_isoestaskpack(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy
single_ASN1_type, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_taskpack_main()

int DBVsnacc_to_dbv_taskpack_main(struct DbvEsTaskPackage *psEsTaskPackage, TaskPackage
*taskpack_ptr, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_intunit()
AsnListCount()
DBVsnacc_to_dbv_permissions()
DBVsnacc_to_dbv_octet_string()
Dbvsnacc_to_dbv_diagrec_list()
DBVsnacc_to_dbv_external()

int DBVsnacc_to_dbv_permissions(DbvPermissions **ppsPermissions, Permissions *permissions, jmp_buf
env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_permisnode()

int DBVsnacc_to_dbv_permisnode(DbvPermissions *psPermissions, PermissionsSeq *data_ptr, jmp_buf env)
DBVsnacc_to_dbv_inter_string()
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
Dbvsnacc_to_dbv_allowfuncts()

int Dbvsnacc_to_dbv_allowfuncts(DbvPermAllowableFunctions **eAllowableFunctions, PermissionsSeqSeqOf
*allowableFunctions, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()

int Dbvsnacc_to_dbv_diagrec_list(DbvDiagRec **dbv_diaglist, AsnList *snacc_diaglist, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_diag_rec()

int DBVdbv_to_snacc_update(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
DBVdbv_to_snacc_update_main()

int DBVdbv_to_snacc_isoupdate(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
DBVdbv_to_snacc_update_main()

int DBVdbv_to_snacc_update_main(Update *update_ptr, struct DbvUpdateExtendedService
*psUpdateExtendedService, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_update_esreq()
DBVdbv_to_snacc_update_taspack()

int DBVdbv_to_snacc_update_esreq(struct UpdateSeq* esRequest, DbvUpdateRequestPackage
*psUpdateRequest, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_parttokeep5()
AsnListNew()
DBVdbv_to_snacc_partnottokeep5()
```

```
int DBVdbv_to_snacc_parttokeep5(struct OriginPartToKeep5 *toKeep, DbvOriginPartToKeep
*psOriginPartToKeep, jmp_buf env)
DBVdbv_to_snacc_inter_string()
ASN_MakeOid()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_partnottokeep5(OriginPartNotToKeep5 *notToKeep, DbvSuppliedRecord
**ppsSuppliedRecord, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_supplied_rec()

int DBVdbv_to_snacc_supplied_rec(SuppliedRecordsSeq *data_ptr, DbvSuppliedRecord *psSuppliedRecord,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_recordid()
DBVdbv_to_snacc_supple_id()
DBVdbv_to_snacc_correl_info()
DBVdbv_to_snacc_external()

int DBVdbv_to_snacc_recordid(struct SuppliedRecordsSeqChoice* recordId, DbvUpdtRecordId
*psUpdtRecordId, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_octet_string()

int DBVdbv_to_snacc_supple_id(struct SuppliedRecordsSeqChoice1 *supplementalId,
DbvUpdtSupplementalId *psUpdtSupplementalId, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_external()

int DBVdbv_to_snacc_correl_info(struct CorrelationInfo* correlationInfo, DbvCorrelationInfo
*psCorrelationInfo, jmp_buf env)
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_update_taspack(struct UpdateSeq1 *taskPackage, DbvUpdateTaskPackage
*psUpdateResponse, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_parttokeep5()
DBVdbv_to_snacc_targetpart()

int DBVdbv_to_snacc_targetpart(struct TargetPart5* targetPart, DbvUpdateTargetPart *psTargetPart,
jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_diagrec_list()
DBVdbv_to_snacc_taskpackrecs()

int DBVdbv_to_snacc_taskpackrecs(TargetPart5SeqOf1 *taskPackageRecords, DbvTaskPackageRecord
**ppsTaskPackageRecord, jmp_buf env)
AsnListAppend()
Asn1Alloc
CheckAsn1Alloc()
DBVdbv_to_snacc_taskpackrec()

int DBVdbv_to_snacc_taskpackrec(TaskPackageRecordStructure *data_ptr, DbvTaskPackageRecord
*psTaskPackageRecord, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_recorsur()
DBVdbv_to_snacc_correl_info()


int DBVdbv_to_snacc_recorsur(struct TaskPackageRecordStructureChoice* recordOrSurDiag,
DbvUpdtRecordOrDiag *psUpdtRecordOrDiag, jmp_buf env)
Asn1Alloc()
```

```
CheckAsn1Alloc()
DBVdbv_to_snacc_external()
DBVdbv_to_snacc_diag_rec()

int DBVsnacc_to_dbv_update(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_update_main()

int DBVsnacc_to_dbv_isoupdate(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_update_main()

int DBVsnacc_to_dbv_update_main(struct DbvUpdateExtendedService *psUpdateExtendedService, Update
*update_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_updatereq()
DBVsnacc_to_dbv_updatersp()

int DBVsnacc_to_dbv_updatereq(DbvUpdateRequestPackage *psUpdateRequest, struct UpdateSeq *esRequest,
jmp_buf env)
DBVsnacc_to_dbv_parttokeep5()
AsnListCount()
CheckDbvAlloc()
DBVsnacc_to_dbv_notokeep()

int DBVsnacc_to_dbv_parttokeep5(DbvOriginPartToKeep *psOriginPartToKeep, struct OriginPartToKeep5*
toKeep, jmp_buf env)
DBVsnacc_to_dbv_inter_string()
DbvAlloc()
CheckDbvAlloc()

int DBVsnacc_to_dbv_notokeep(DbvSuppliedRecord **ppsSuppliedRecord, OriginPartNotToKeep5 *notToKeep,
jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_supp_rec()

int DBVsnacc_to_dbv_supp_rec(DbvSuppliedRecord *psSuppliedRecord, SuppliedRecordsSeq *data_ptr,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_recordid()
DBVsnacc_to_dbv_suppid()
DBVsnacc_to_dbv_corr_info()
DBVsnacc_to_dbv_external()

int DBVsnacc_to_dbv_recordid(DbvUpdtRecordId *psUpdtRecordId, struct SuppliedRecordsSeqChoice*
recordId, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_octet_string()

int DBVsnacc_to_dbv_suppid(DbvUpdtSupplementalId *psUpdtSupplementalId, struct
SuppliedRecordsSeqChoice1* supplementalId, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_external()

int DBVsnacc_to_dbv_corr_info(DbvCorrelationInfo *psCorrelationInfo, struct CorrelationInfo*
correlationInfo, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_updatersp(DbvUpdateTaskPackage *psUpdateResponse, struct UpdateSeq1
*taskPackage, jmp_buf env)
```

```
DBVsnacc_to_dbv_parttokeep5()
DBVsnacc_to_dbv_targetPart()

int DBVsnacc_to_dbv_targetPart(DbvUpdateTargetPart *psTargetPart, struct TargetPart5* targetPart,
jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
Dbvsnacc_to_dbv_diagrec_list()
AsnListCount()
DBVsnacc_to_dbv_taskpack()

int DBVsnacc_to_dbv_taskpack(DbvTaskPackageRecord **ppsTaskPackageRecord, TargetPart5SeqOf1
*taskPackageRecords, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_taskpackrec()

int DBVsnacc_to_dbv_taskpackrec(DbvTaskPackageRecord *psTaskPackageRecord,
TaskPackageRecordStructure *data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_recordiag()
DBVsnacc_to_dbv_corr_info()

int DBVsnacc_to_dbv_recordiag(DbvUpdtRecordOrDiag *psUpdtRecordOrDiag, struct
TaskPackageRecordStructureChoice* recordOrSurDiag, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()
DBVsnacc_to_dbv_diag_rec()

int DBVsnacc_to_dbv_unknown(DbvSingleASN1Type *psSingleASN1Type, AsnAnyDefinedBy single_ASN1_type,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVdbv_to_snacc_aligned(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf env)
DBVdbv_to_snacc_octet_string()

int DBVsnacc_to_dbv_aligned(DbvASN1External *dbv_external, NEWEXTERNAL *snacc_external, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()

int DBVdbv_to_snacc_arbitrary(NEWEXTERNAL *snacc_external, DbvASN1External *dbv_external, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
SetAsnBit()
ClrAsnBit()

int DBVsnacc_to_dbv_arbitrary(DbvASN1External *dbv_external, NEWEXTERNAL *snacc_external, jmp_buf
env)
GetAsnBit()
DbvAlloc()
CheckDbvAlloc()
```

Checking:

```
int DBVcheck_dbv_external(DbvAssocId lAssocId, DbvASN1External *psDbvASN1External, char
*error_buffer, DbvError **ppsSIError)
DBVcheck_dbv_recsyn()
DBVcheck_dbv_allowed_external()
DBVdbv_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_singleasn1()
DBVcheck_dbv_arbitrary()
DBVdbv_invalid_enum_error()
```

```
int DBVcheck_dbv_singleasn1(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVcheck_at_unstructuredtext()
DBVcheck_at_opacrecord()
DBVcheck_at_summaryrecord()
DBVcheck_at_resourcereport1()
DBVcheck_at_resourcereport2()
DBVcheck_at_estaskpackage()
DBVcheck_at_updateextendedservice()
sa_strcat()
DBVdbv_invalid_enum_error()

int DBVcheck_at_unstructuredtext(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char
*oid, char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()

int DBVcheck_at_opacrecord(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_opac()

int DBVcheck_dbv_opac(DbvAssocId lAssocId, struct DbvOpacRecord *psOpacRecord, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_external()
DBVdbv_field_missing_error()
DBVcheck_dbv_holdrec()

int DBVcheck_dbv_holdrec(DbvAssocId lAssocId, DbvHoldingsRecord *psHoldingsData, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_external()
DBVcheck_dbv_holdandcirc()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_holdandcirc(DbvAssocId lAssocId, DbvHoldingsAndCircData *psHoldingsAndCirc, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVcheck_dbv_volume()
DBV_check_dbv_circrec()

int DBVcheck_dbv_volume(DbvAssocId lAssocId, DbvVolume **ppsVolume, char *error_buffer, DbvError
**ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()

int DBV_check_dbv_circrec(DbvAssocId lAssocId, DbvCircRecord **ppsCircRecord, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_circrec()

int DBVcheck_circrec(DbvAssocId lAssocId, DbvCircRecord *psCircRecord, char *error_buffer, DbvError
**ppsSIError)
sa_strcpy()
DBVcheck_dbv_bool()
sa_strcat()
DBVdbv_bool_error()

int DBVcheck_at_summaryrecord(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
```

```
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_summary()

int DBVcheck_dbv_summary(DbvAssocId lAssocId, struct DbvSummaryRecord *psSummaryRecord, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_otherinfo()

int DBVcheck_at_resourcereport1(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_resrep1()

int DBVcheck_dbv_resrep1(DbvAssocId lAssocId, struct DbvResourceReport1 *psResourceReport1, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_estimate1()

int DBVcheck_dbv_estimate1(DbvAssocId lAssocId, DbvEstimate1 *psEstimate, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_at_resourcereport2(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_resrep2()

int DBVcheck_dbv_resrep2(DbvAssocId lAssocId, struct DbvResourceReport2 *psResourceReport2, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_estimate2()

int DBVcheck_dbv_estimate2(DbvAssocId lAssocId, DbvEstimate2 *psEstimate, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_strornum()
DBVcheck_dbv_intunit()

int DBVcheck_at_estaskpackage(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char *oid,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_estask()

int DBVcheck_dbv_estask(DbvAssocId lAssocId, struct DbvEsTaskPackage *psEsTaskPackage, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVcheck_dbv_validoid()
sa_strcat()
DBVdbv_oid_error()
DBVcheck_dbv_intunit()
DBVdbv_field_missing_error()
DBVcheck_dbv_allowfuncs()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()
```

```
DBVcheck_dbv_multdiag()
DBVcheck_dbv_external()

int DBVcheck_at_updateextendedservice(DbvAssocId lAssocId, DbvSingleASN1Type *psSingleASN1Type, char
*oid, char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_external_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_update()

int DBVcheck_dbv_update(DbvAssocId lAssocId, struct DbvUpdateExtendedService
*psUpdateExtendedService, char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_updatereq()
DBVcheck_dbv_updatersp()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_updatereq(DbvAssocId lAssocId, DbvUpdateRequestPackage *psUpdateRequest, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVdbv_field_missing_error()
sa_strcat()
DBVcheck_dbv_originpart()
DBVcheck_dbv_suppliedrecs()

int DBVcheck_dbv_originpart(DbvAssocId lAssocId, DbvOriginPartToKeep *psOriginPartToKeep, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVcheck_dbv_enum()
sa_strcat()
DBVdbv_invalid_enum_error()
DBVcheck_dbv_validoid()
DBVdbv_oid_error()

int DBVcheck_dbv_suppliedrecs(DbvAssocId lAssocId, DbvSuppliedRecord **ppsSuppliedRecord, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_supprec()

int DBVcheck_dbv_supprec(DbvAssocId lAssocId, DbvSuppliedRecord *psSuppliedRecord, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_recid()
DBVcheck_dbv_suppid()
DBVcheck_dbv_corrinfo()
DBVcheck_dbv_external()

int DBVcheck_dbv_recid(DbvAssocId lAssocId, DbvUpdtRecordId *psUpdtRecordId, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_suppid(DbvAssocId lAssocId, DbvUpdtSupplementalId *psUpdtSupplementalId, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_corrinfo(DbvAssocId lAssocId, DbvCorrelationInfo *psCorrelationInfo, char
*error_buffer, DbvError **ppsSIError)

int DBVcheck_dbv_updatersp(DbvAssocId lAssocId, DbvUpdateTaskPackage *psUpdateResponse, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVdbv_field_missing_error()
```

```
sa_strcat()
DBVcheck_dbv_originpart()
DBVcheck_dbv_targetpart()

int DBVcheck_dbv_targetpart(DbvAssocId lAssocId, DbvUpdateTargetPart *psTargetPart, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
DBVcheck_dbv_enum()
sa_strcat()
DBVdbv_invalid_enum_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_multdiag()
DBVcheck_dbv_taskpackrec()

int DBVcheck_dbv_taskpackrec(DbvAssocId lAssocId, DbvTaskPackageRecord *psTaskPackageRecord, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_recordiag()
DBVcheck_dbv_corrinfo()
DBVcheck_dbv_enum()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_recordiag(DbvAssocId lAssocId, DbvUpdtRecordOrDiag *psUpdtRecordOrDiag, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVcheck_dbv_surrdiag()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_arbitrary(DbvAssocId lAssocId, DbvOctetString *psArbitrary, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
DBVcheck_bit_string()
sa_strcat()
DBVdbv_bit_string_error()
```

z3950_shared.c (routines shared by origin and target)

```
void DBVstring_to_up_case(char *input)

void DBVget_token_from_string(char *token_obtained, char *token_delimiter, char *string)

int DBVdbv_to_snacc_octet_string(AsnOcts *snacc_asnocts, DbvOctetString *dbv_octet_string, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_inter_string(InternationalString *snacc_interstr, DbvInternationalString
*dbv_interstr, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()

int DBVsnacc_to_dbv_octet_string(DbvOctetString *dbv_octet_string, AsnOcts *snacc_asnocts, jmp_buf
env)
DbvAlloc()
CheckDbvAlloc()

int DBVsnacc_to_dbv_inter_string(DbvInternationalString *dbv_interstr, InternationalString
*snacc_interstr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()

int DBVcheck_bit_string(char  *bit_string)

int DBVcheck_chosen_options(char  *bit_string)
```

```
int DBVdbv_to_snacc_el_set_names(struct ElementSetNames* simple, DbvElementSetNames
*psElementSetNames, jmp_buf env)
DBVdbv_to_snacc_el_generic()
DBVdbv_to_snacc_el_specific()

int DBVdbv_to_snacc_el_generic(struct ElementSetNames* simple, DbvElementSetNames
*psElementSetNames, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_el_specific(struct ElementSetNames* simple, DbvElementSetNames
*psElementSetNames, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
AsnListAppend()
Asn1Alloc()
DBVdbv_to_snacc_inter_string()


int DBVdbv_to_snacc_records(struct Records* records, DbvRecords *psRecords, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_resprec()
DBVdbv_to_snacc_defaultformat()
DBVdbv_to_snacc_multdiag()


int DBVdbv_to_snacc_resprec(RecordsSeqOf *responseRecords, DbvNamePlusRecord **ppsResponseRecord,
jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_nameplusrec()


int DBVdbv_to_snacc_nameplusrec(NamePlusRecord *data_ptr, DbvNamePlusRecord *psResponseRecord,
jmp_buf env)
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()
DBVdbv_to_snacc_diag_rec()
DBVdbv_to_snacc_fragsyntax()


int DBVdbv_to_snacc_diag_rec(struct DiagRec *surrogateDiagnostic, DbvDiagRec *psSurrogateDiagnostic,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_defaultformat()
DBVdbv_to_snacc_external()


int DBVdbv_to_snacc_defaultformat(struct DefaultDiagFormat* defaultFormat, DbvDefaultDiagFormat
*psDefault, jmp_buf env)
ASN_MakeOid()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_addinfo()


int DBVdbv_to_snacc_addinfo(struct DefaultDiagFormatChoice* addinfo, DbvAdditionalInfo
sAdditionalInfo, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()


int DBVdbv_to_snacc_fragsyntax(struct FragmentSyntax *snaccfragment, DbvFragmentSyntax *dbvfragment,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_external()
```

```
DBVdbv_to_snacc_octet_string()


int DBVdbv_to_snacc_multdiag(RecordsSeqOf1 *multipleNonSurDiagnostics, DbvDiagRec
**ppsMultipleNonSurrogateDiag, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_diag_rec()

DBVsnacc_to_dbv_records(DbvRecords *psRecords, struct Records* records, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_resprec()
DBVsnacc_to_dbv_defaultformat()
DBVsnacc_to_dbv_multdiag()


int DBVsnacc_to_dbv_resprec(DbvNamePlusRecord **ppsResponseRecord, RecordsSeqOf *responseRecords,
jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_nameplusrec()

int DBVsnacc_to_dbv_nameplusrec(DbvNamePlusRecord *psResponseRecord, NamePlusRecord *data_ptr,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_external()
DBVsnacc_to_dbv_diag_rec()
DBVsnacc_to_dbv_fragsyntax()


int DBVsnacc_to_dbv_diag_rec(DbvDiagRec *psSurrogateDiagnostic, struct DiagRec *surrogateDiagnostic,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_defaultformat()
DBVsnacc_to_dbv_external()


int DBVsnacc_to_dbv_defaultformat(DbvDefaultDiagFormat *psDefault, struct DefaultDiagFormat*
defaultFormat, jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_addinfo()


int DBVsnacc_to_dbv_addinfo(DbvAdditionalInfo *psAdditionalInfo, struct DefaultDiagFormatChoice*
addinfo, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_fragsyntax(DbvFragmentSyntax *dbvfragment, struct FragmentSyntax* snaccfragment,
jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_external()
DBVsnacc_to_dbv_octet_string()

int DBVsnacc_to_dbv_multdiag(DbvDiagRec **ppsMultipleNonSurrogateDiag, RecordsSeqOf1
*multipleNonSurDiagnostics, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_diag_rec()


int DBVcheck_dbv_enum(unsigned long field, unsigned long bottom_val, unsigned long top_val)
```

```
int DBVdbv_field_missing_error(DbvAssocId lAssocId, char *field_in_error, DbvError **ppsSIError)
sa_strcpy()
DBVfill_error_structure()


int DBVdbv_bit_string_error(DbvAssocId lAssocId, char *field_in_error, char *value, DbvError
**ppsSIError)
sa_strcpy()
sa_clear_array()
DBVfill_error_structure()

int DBVdbv_options_chosen_error(DbvAssocId lAssocId, char *field_in_error, char *value, DbvError
**ppsSIError)
sa_strcpy()
sa_clear_array()
DBVfill_error_structure()

int DBVdbv_invalid_enum_error(DbvAssocId lAssocId, char *field_in_error, DbvError **ppsSIError)
sa_strcpy()
DBVfill_error_structure()

int DBVdbv_bool_error(DbvAssocId lAssocId, char *field_in_error, DbvError **ppsSIError)
sa_strcpy()
DBVfill_error_structure()

int DBVdbv_oid_error(DbvAssocId lAssocId, char *field_in_error, DbvError **ppsSIError)
sa_strcpy()
DBVfill_error_structure()

int DBVdbv_external_oid_error(DbvAssocId lAssocId, char *oid, char *field_in_error, DbvError
**ppsSIError)
sa_strcpy()
sa_clear_array()
DBVfill_error_structure()

int DBVcheck_dbv_bool(DbvBool boolval)

int DBVcheck_dbv_recsyn(char *oid)

int DBVcheck_dbv_allowed_external(char *oid)

int DBVcheck_dbv_attsetoid(char *oid)

int DBVcheck_dbv_diagsetoid(char *oid)

int DBVcheck_dbv_resrepoid(char *oid)

int DBVcheck_dbv_updateoid(char *oid)

int DBVcheck_dbv_validoid(char *oid)

int DBVcheck_generic(DbvAssocId lAssocId, DbvInternationalString *psGeneric, DbvError **ppsSIError,
char *name)
DBVdbv_field_missing_error()

int DBVcheck_dbspecific(DbvAssocId lAssocId, DbvElementSetsDbSpecific **ppsElementSetsDbSpecific,
DbvError **ppsSIError, char *name)
DBVdbv_field_missing_error()

int DBVcheck_dbv_records(DbvAssocId lAssocId, DbvRecords *psRecords, char *error_buffer, DbvError
**ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_response_rec()
DBVcheck_dbv_default()
DBVcheck_dbv_multdiag()
DBVdbv_invalid_enum_error()

int DBVcheck_response_rec(DbvAssocId lAssocId, DbvNamePlusRecord **ppsResponseRecord, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
```

```
DBVcheck_dbv_nameplusrec()

int DBVcheck_dbv_nameplusrec(DbvAssocId lAssocId, DbvNamePlusRecord *psResponseRecord, char
*error_buffer, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVcheck_dbv_surrdiag()
DBVcheck_dbv_fragment()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_surrdiag(DbvAssocId lAssocId, DbvDiagRec *psSurrogateDiagnostic, char
*error_buffer, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_default()
DBVcheck_dbv_external()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_default(DbvAssocId lAssocId, DbvDefaultDiagFormat *psDefault, char *error_buffer,
DbvError **ppsSIError)
DBVcheck_dbv_validoid()
DBVdbv_oid_error()
DBVdbv_field_missing_error()
DBVdbv_invalid_enum_error()


int DBVcheck_dbv_fragment(DbvAssocId lAssocId, DbvFragmentSyntax *psFragment, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVdbv_invalid_enum_error()

int DBVcheck_dbv_multdiag(DbvAssocId lAssocId, DbvDiagRec **ppsMultipleNonSurrogateDiag, char
*error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_surrdiag()

int DBVdbv_to_snacc_strornum(StringOrNumeric *data_ptr, DbvStringOrNumeric *psAttributeElement,
jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()

int DBVdbv_to_snacc_intunit(struct IntUnit* integerAndUnit, DbvIntUnit *psIntegerAndUnit, jmp_buf
env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_unit()

int DBVdbv_to_snacc_unit(struct Unit* unitUsed, DbvUnit sUnitUsed, jmp_buf env)
DBVdbv_to_snacc_inter_string()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_strornum()

int DBVsnacc_to_dbv_strornum(DbvStringOrNumeric *psList, StringOrNumeric *data_ptr, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()

int DBVsnacc_to_dbv_intunit(DbvIntUnit *psIntegerAndUnit, struct IntUnit* integerAndUnit, jmp_buf
env)
DBVsnacc_to_dbv_unit()


int DBVsnacc_to_dbv_unit(DbvUnit *psUnitUsed, struct Unit* unitUsed, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_strornum()
```

```
int DBVdbv_to_snacc_attrterm(struct AttributesPlusTerm* attrTerm, DbvAttributesPlusTerm
*psAttributesPlusTerm, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_attribute_list()
Asn1Alloc()
DBVdbv_to_snacc_term()


int DBVdbv_to_snacc_attribute_list(AttributeList* attributes, DbvAttributeElement
**ppsAttributeElement, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_attributes()

int DBVdbv_to_snacc_attributes(AttributeElement *data_ptr, DbvAttributeElement *psAttributeElement,
jmp_buf env)
ASN_MakeOid()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_attval()

int DBVdbv_to_snacc_attval(struct AttributeElementChoice* attributeValue, DbvAttributeValue
sAttributeValue, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_attcomplex()

int DBVdbv_to_snacc_attcomplex(struct AttributeElementChoiceSeq* complex, DbvComplexValue
*psComplexValue, jmp_buf env)
AsnListNew()
CheckAsn1Alloc()
DBVdbv_to_snacc_attcomplist()
DBVdbv_to_snacc_attcompsem()

int DBVdbv_to_snacc_attcomplist(AttributeElementChoiceSeqSeqOf* list, DbvStringOrNumeric **ppsList,
jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_strornum()

int DBVdbv_to_snacc_attcompsem(AttributeElementChoiceSeqSeqOf1* semanticAction, Ulong
**ppulSemanticAction, jmp_buf env)
AsnListAppend()
Asn1Alloc()
CheckAsn1Alloc()

int DBVdbv_to_snacc_term(struct Term* term, DbvTerm sTerm, jmp_buf env)
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_octet_string()
DBVdbv_to_snacc_inter_string()
ASN_MakeOid()
DBVdbv_to_snacc_external()
DBVdbv_to_snacc_intunit()


int DBVsnacc_to_dbv_attrterm(DbvAttributesPlusTerm *psAttributesPlusTerm, struct AttributesPlusTerm*
attrTerm, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
FOR_EACH_LIST_ELMT()
DBVsnacc_to_dbv_attributes()
DBVsnacc_to_dbv_term()


int DBVsnacc_to_dbv_attributes(DbvAttributeElement *psAttributeElement, AttributeElement *data_ptr,
jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
```

```
DBVsnacc_to_dbv_attvalue()


int DBVsnacc_to_dbv_attvalue(DbvAttributeValue *psAttributeValue, struct AttributeElementChoice*
attributeValue, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_complex()

int DBVsnacc_to_dbv_complex(DbvComplexValue *psComplexValue, struct AttributeElementChoiceSeq*
complex, jmp_buf env)
AsnListCount()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_list()
DBVsnacc_to_dbv_semantic()


int DBVsnacc_to_dbv_list(DbvStringOrNumeric **ppsList, AttributeElementChoiceSeqSeqOf* list, jmp_buf
env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_strornum()

int DBVsnacc_to_dbv_semantic(Ulong **ppulSemanticAction, AttributeElementChoiceSeqSeqOf1*
semanticAction, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()

int DBVsnacc_to_dbv_term(DbvTerm *psTerm, struct Term* term, jmp_buf env)
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_octet_string()
DBVsnacc_to_dbv_inter_string()
ASN_EncodedToString()
DBVsnacc_to_dbv_external()
DBVsnacc_to_dbv_intunit()

int DBVsnacc_to_dbv_attribute_list(DbvAttributeElement **ppsAttributeElement, AttributeList
*attributes, jmp_buf env)
FOR_EACH_LIST_ELMT()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_attributes()


int DBVdbv_to_snacc_specification(struct Specification* generic, DbvSpecification *psGeneric,
jmp_buf env)
ASN_MakeOid()
Asn1Alloc()
CheckAsn1Alloc()
DBVdbv_to_snacc_inter_string()
DBVdbv_to_snacc_external()

int DBVsnacc_to_dbv_specification(DbvSpecification *psGeneric, struct Specification* generic,
jmp_buf env)
ASN_EncodedToString()
DbvAlloc()
CheckDbvAlloc()
DBVsnacc_to_dbv_inter_string()
DBVsnacc_to_dbv_external()

char *DBVgobble_up_memory()

int DBVcheck_attsplusterm(DbvAssocId lAssocId, DbvAttributesPlusTerm *psAttributesPlusTerm, DbvError
**ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_attelementI()
DBVcheck_term()
```

```
int DBVcheck_attelement(DbvAssocId lAssocId, DbvAttributeElement *psAttributeElement, DbvError
**ppsSIError)
DBVcheck_dbv_validoid()
DBVdbv_field_missing_error()
DBVcheck_attval()

int DBVcheck_attval(DbvAssocId lAssocId, DbvAttributeValue *psAttributeValue, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_complexval()
DBVdbv_invalid_enum_error()

int DBVcheck_complexval(DbvAssocId lAssocId, DbvComplexValue *psComplexValue, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_strornum()

int DBVcheck_dbv_strornum(DbvAssocId lAssocId, DbvStringOrNumeric *psList, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error()
DBVdbv_invalid_enum_error()

int DBVcheck_term(DbvAssocId lAssocId, DbvTerm *psTerm, DbvError **ppsSIError)
DBVdbv_field_missing_error()
DBVcheck_dbv_validoid()
DBVdbv_oid_error()
DBVcheck_dbv_external()
DBVcheck_dbv_intunit()
DBVdbv_invalid_enum_error()


int DBVcheck_dbv_intunit(DbvAssocId lAssocId, DbvIntUnit *psIntegerAndUnit, char *error_buffer,
DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVcheck_dbv_strornum()

int DBVcheck_specification(DbvAssocId lAssocId, DbvSpecification *psGeneric, DbvError **ppsSIError)
DBVcheck_dbv_recsyn()
DBVdbv_oid_error()
DBVdbv_field_missing_error()
DBVcheck_dbv_external()
DBVdbv_invalid_enum_error()


int DBVcheck_dbv_allowfuncs(DbvAssocId lAssocId, DbvPermAllowableFunctions **eAllowableFunctions,
char *error_buffer, DbvError **ppsSIError)
sa_strcpy()
sa_strcat()
DBVdbv_field_missing_error
DBVcheck_dbv_
DBVdbv_invalid_enum_error()

int sa_clear_array(char *array)
int sa_strcat(char *dest, char *src)
int sa_strcpy(char *dest, char *src)
```

z3950_send_receive.c:

```
int DBVencode_and_send(PDU *PDU_ptr, DbvAssocId lAssocId, DbvError **ppsError)
DBVEnterFuncTrace()
DBVAllocCheck()
DBVAlloc()
DBV_reset_external_counters()
DBVTraceLevel()
DBVTrace()
DBVencode_pdu_to_bytes()
SIWhichStack()
SL_Current()
```

```
DBVsend_data_over_socket()
DBVLeaveFuncTrace()
```

z3950_enc_dec.c:

```
int DBVencode_pdu_to_bytes(DbvAssocId lAssocId, PDU *pdu_ptr, char **encoded_pdu, long
*encoded_pdu_length, DbvError **ppsSIError)
DBVinit_external_data()
SBufInit()
SBufResetInWriteRvsMode()
BEncPDU()
SBufWriteError()
DBVfill_error_structure()
FreePDU()
```

z3950_send_receive.c:

```
int DBVsend_data_over_socket(int socketfd, char *data_start, int length)
```

macros called:

```
src/headers/dbvproto.h:
#define DbvAlloc(size)  calloc(1,size)
#define CheckDbvAlloc(ptr, env) \
  if ((ptr) == NULL) \
        longjmp(env, -1)
```

No system calls (e.g. sprintf) or ISODE function calls shown.

snacc generated file functions called:

z39_50_apdu_1994.c:

```
AsnLen BEncPDU PARAMS((b, v),
BUF_TYPE b _AND_
PDU* v)

void BDecPDU PARAMS((b, result, bytesDecoded, env),
BUF_TYPE b _AND_
PDU* result _AND_
AsnLen* bytesDecoded _AND_
ENV_TYPE env)

void FreePDU PARAMS((v),
PDU* v)
```

The tree of functions below BEncPDU, BDecPDU and FreePDU are not shown.

z39_50_externals_a_1994.c:

```
void InitAnyZ39_50_EXTERNALS_A_1994()
```

z39_50_externals_b_1994.c:

```
void InitAnyZ39_50_EXTERNALS_B_1994()
```

z39_50_externals_d_1994.c:

```
void InitAnyZ39_50_EXTERNALS_D_1994()
```

z39_50_externals_e_1994.c:

```
void InitAnyZ39_50_EXTERNALS_E_1994()
```

z39_50_externals_f_1994.c:

```
void InitAnyZ39_50_EXTERNALS_F_1994()
```

z39_50_externals_iso_1994.c:

```
void InitAnyZ39_50_EXTERNALS_ISO_1994()
```

snacc library functions called:

src/snacc/libs/asn_bits.h:

```
void SetAsnBit PROTO((AsnBits* b1, unsigned long int bit))

void ClrAsnBit PROTO((AsnBits* b1, unsigned long int bit));

int GetAsnBit PROTO((AsnBits* b1, unsigned long int bit));
```

snacc/libs/sbuf.h:

```
#define SBufInit(b, data, dataLen)\
{ (b)->readError = (b)->writeError = 1;\
  (b)->blkStart = data;\
  (b)->blkEnd = data + dataLen;\
  (b)->dataStart = (b)->dataEnd = (b)->readLoc = (b)->blkEnd;\
}


#define SBufResetInWriteRvsMode(b)\
{ (b)->dataStart = (b)->dataEnd = (b)->blkEnd;\
  (b)->writeError = 0;\
  (b)->readError = 1;\
}

#define SBufInstallData(b, data, dataLen)\
   SBufInit(b, data, dataLen);\
   (b)->dataStart = (b)->blkStart;\
   SBufResetInReadMode(b);


#define SBufWriteError(b)\
  ((b)->writeError)
```

Note: No supporting functions/structures shown

snacc/libs/asn_oid.c:

```
AsnLen EncodedOidLen PARAMS((oid),
OID_SNACC* oid)
```

snacc/libs/asn_config.h

```
#define Asn1Alloc(size) (void *) calloc(1, size)
#define Asn1Free(ptr)   free(ptr)
#define CheckAsn1Alloc(ptr, env)\
   if ((ptr) == NULL)\
      longjmp(env, -27);
```

snacc/libs/asn_list.c:

```
AsnList* AsnListNew PARAMS( (dataSize),
int dataSize )

void* AsnListAppend PARAMS( (list),
AsnList* list )

int AsnListCount PARAMS( (list),
AsnList* list )
```

snacc/libs/asn_list.h:

```
#define  FOR_EACH_LIST_ELMT( elmt,  al)\
if ((al) != NULL)\
    for ((al)->curr = (al)->first;\
         ((al)->curr != NULL) && (((elmt) =(void*)(al)->curr->data) != NULL)
;\
         (al)->curr = (al)->curr->next )
```

11 GLOBAL VARIABLE AND STRUCTURE INFORMATION

All system global variables will be prefixed with capital DBV. Variable name conflicts can be avoided by not using this prefix.

11.1 Origin

`char *ppcDBVProposedPresContexts[NPCTX];` – Stores the proposed presentation context definition list OID's in char * format so that they can be compared with the negotiated list in the AAsociate.Confirmation service primitive. NPCTX is defined in the Isode header "psap.h" and is the maximum number of contexts allowed in the list.

11.2 Target

`int DBVSocket` – This variable is used to store the incoming socket file descriptor. It is currently parsed from the command line and placed here, so it can be accessed in the `ReceiveAssociateRequest()` function.

`char DBVAddress[64]` – Stores the address of the origin system connecting. It is placed here in `TargetInitialize()` so it can be accessed in the `ReceiveAssociateRequest()` function.

`OsiOrInternetAddress DBVOsiOrInternetAddress` – Used to store whether the target will be using OSI or TCP/IP. This is used to configure the target inside `ReceiveAssociateRequest()`.

`struct AcSAPstart *DBVpsAssocControlStart=NULL` – Used in the target to pass details from `DbvTargetInitialise()` to `DbvReceiveAssociateRequest()`.

11.3 Common

`SL DBVAssociateList` – For both the origin and the target a linked list of associations is stored. This is used for many reasons such as making sure that association Id's supplied from above the API are valid and calculating which stack the association is using.

Although a target application will only ever have one association, it is still place in a linked list so that the same utilitiy functions can be used by both the origin and target.

`SL DBVTraceList` – This linked list is used to store the trace information associated with associations. This variables use is limited to the public and internal trace functions inside si_utilities.c

`char DBVErrorBuffer[SA_ARRAY_SIZE]` – Used to construct error messages and store error messages before error structure is filled out.

`char DBVTraceBuffer[128]` – Used to construct messages before they are wrtten to the trace file.

`char DBVACSEFileName[128] = DBV_ACSE_FILE` – Holds the current ACSE configuration filename. Defaults to DBV_ACSE_FILE ("./ACSE_Configration_File" - See file **sil_defs.h**).

```
typedef enum DBVOriginOrTarget {
      O = 1, T = 2
} DBVOriginOrTarget;
```

`DBVAssociateDetails` stores details about connections which have been initiated/completed:

```
typedef struct DBVAssociateDetails {
      DbvAssocId lAssocId;
      DbvOsiOrInternetAddress eOsiOrInternetAddress;
      DbvFds ulTcpSocket;
      DbvAssociateState eAssociateState;
      DbvReleaseState eReleaseState;
      struct sockaddr_in sa;
      long lCurrentState;     /* Stores the current state of the association.
                        Used (obviously) in the state machine */
#ifndef NO_OSI
      long lAssocResult;      /* lAssocResult stores the ISODE value for the
                        association state:  OK/DONE, NOTOK,
                        CONNECTING_1, CONNECTING_2 */

      long lOSIAssocId;       /* lOSIAssocId contains the association
                        ID which ISODE assigns the association */
      struct AcSAPstart *psAssocControlStart; /* Used to transfer information
                        from DbvReceiveAssociateRequest to
                        DbvAssociateResponse */
#endif
      DbvApplicationProtocol eApplicationProtocol; /* Used between
                        DbvReceiveAssociateRequest and
                        DbvAssociateResponse */

      DBVOriginOrTarget eOriginOrTarget;
      union {
            PDUOriginInformation sPDUOriginInformation;
            PDUTargetInformation sPDUTargetInformation;
      } uOOrT; /* Origin or Target Association */
} DBVAssociateDetails;
```

Store the relevant trace details for Associations:

```
typedef struct DBVTraceDetails {
        DbvTraceLevel DBVTraceLevel;
        FILE *DBVTraceFile;
        DbvAssocId lAssocId;
} DBVTraceDetails;
```

12 ERROR STRUCTURES, CODES AND EXPLANATIONS

12.1 Error Structures

```
typedef struct DbvError {
  DbvErrorClass               eSIErrorClass;
  union {
    DbvErrorCode               lIfOsCoError;
    DbvAbort           *psSIAbort;
  }                    uErrorCode;
  DbvAssociationState  eAssociationState;
} DbvError;

typedef enum DbvErrorClass {
               SI_Communication_Error = 0,
               SI_Interface_Error = 1,
               SI_OperatingSystem_Error = 2,
               SI_Abort = 3
} DbvErrorClass;

typedef long   DbvErrorCode;

typedef struct DbvAbort {
  DbvAbortSource       eAbortSource;
  DbvAbortReason       lAbortReason;
  char                 *pcDiagnostics;
} DbvAbort;

/* Association state upon return of an API function: */
typedef enum DbvAssociationState {
               Association_Open = 0,
               Association_Closed = 1,
               Association_Unknown_State = 2
} DbvAssociationState;

typedef enum DbvAbortSource {
               SI_ACSE_User = 0,
               SI_ACSE_Provider = 1,
               SI_Presentation_User = 2,
               SI_Presentation_Provider = 3,
               SI_Transport_Provider = 4
} DbvAbortSource;
typedef long   DbvAbortReason;
```

Whenever an error occurs within an API function, this structure is filled out with diagnostic information. If the error is not an operating system, communication or interface error, then the `DbvAbort` structure will be filled with an error code. Otherwise the `lIfOsCoError` will be completed. In the latter case, an error message will be saved to either the associated trace file or API log file. These codes are defined in the file **dbv_proto.h** and are also listed in Section **12.2** below. If appropriate, the `pcDiagnostics` character array of the `DbvAbort` structure will contain a written description of the error. The basic forms of these explanations can also be found in section 12.2.

Note that Operating system signals are not trapped within the API since such signals should be trapped and handled by the Service User application and any such signals in the API would cause a conflict for this.

12.2 DBV-OSI II Error codes:

Below are a list of the error codes and explanations which will be returned in the error structure `DbvAbort`.

The error codes can be found in the header file **dbv_proto.h**.

The error codes will be categorised as follows:

- **SI_Interface_Error -** Returned if the error is caused by the service-user component.
- **SI_Communication_Error** - Returned if an error is detected in an incoming APDU, an APDU is received out of sequence or a communication function (TCP or OSI) failed.
- **SI_OperatingSystem_Error** - Returned if the operating system reports an error at any time. Also used for some internal errors (occurrences of which will be very unlikely).
- **SI_Abort** - This error is returned if an error has caused the association to be made invalid.

12.2.1 General Error Codes:

Error: DBV_ABORT
Description: The Association has been aborted by the peer. The association will no longer exist.
Number: 0
Message: "The Association has been aborted"
Class: SI_Abort

Error: DBV_INTERRUPT
Description: A call to a system function was interrupted by a signal from the system (e.g. TCP read/write/select returned EINTR). This is not usually terminal.
Number: 1
Message: "A function call was interrupted by the system"
Class: SI_Communication_Error

Error: DBV_ASSOCDEAD
Description: An Association has been unexpectedly lost during an operation on it. This is different from an abort indication or release indication and indicates that the association has been lost for some other reason.
Number: 2
Message: "This association has been unexpectedly lost"
Class: SI_Abort

Error: DBV_INVALIDASSOC
Description: An API function has been called with an invalid association ID.
Number: 3
Message: "The association %d is invalid"
Class: SI_Interface_Error

Error: DBV_INVALIDSTYPE
Description: The Association has an invalid stack type. Most likely indicates an internal error (association details are stored internally in a linked list.

Number: 4
Message: "Association %d has an invalid stack type"
Class: SI_OperatingSystem_Error

Error: DBV_NOASSOC
Description: There are currently no associations
Number: 6
Message: "There are currently no associations"
Class: SI_Interface_Error

Error: DBV_INVALIDPARAMS
Description: The function has failed because one or more of the parameters was invalid
Number: 7
Message: "Function called with invalid parameters.\nDiagnostics: %s"
Class: SI_Interface_Error

Error: DBV_SYSERROR
Description: A Unix function call returned an error
Number: 8
Message: "A call to the system function %s returned error: %s"
Class: SI_OperatingSystem_Error

Error: DBV_INVALIDCALL
Description: A call to the API function was not appropriate.
Number: 9
Message: "A call to this API function is invalid.\nDiagnostics: %s"
Class: SI_Interface_Error

12.2.2 Errors Associated with TCP/IP Connections:

Error: DBV_TCP_CLOSEFAILED
Description: The TCP function close failed for the given association. Receiving this error is very unlikely. Recommendation: If returned (most probably from DbvAbortRequest() DbvReleaseRequest/Response()) treat the association as closed and continue.
Number: 30
Message: "TCP: close() failed for Association %d\nDiagnostics: %s"
Class: SI_Communication_Error

Error: DBV_TCP_CONNECTFAILED
Description: The TCP/IP function connect() failed. Most likely because the specified port number was incorrect. Or address was invalid (if specified in number format e.g. 193.195.22.1)
Number: 31
Message: "TCP: connect() failed.\nDiagnostics :%s"
Class: SI_Communication_Error

Error: DBV_TCP_INVALIDADDRESS
Description: The TCP/IP address specified was invalid. Returned if address cannot be resolved by gethostbyname() function.
Number: 32

Message: "TCP/IP: address %s is invalid"
Class: SI_Communication_Error

Error: DBV_TCP_SOCKETFAILED
Description: A socket could not be created. A call to function socket() failed.
Number: 33
Message: "TCP/IP: could not create socket.\nDiagnostics: %s"
Class: SI_Communication_Error

12.2.3 Errors Associated with OSI Connections:

Error: DBV_OSI_INVALIDADDRESS
Description: The specified OSI address is invalid
Number: 40
Message: "OSI: address %s is invalid"
Class: SI_Interface_Error

Error: DBV_OSI_ISODE_ERROR
Description: An error occurred in an ISODE function. This error should never actually be returned. If it is, then view the diagnostic string to see what went wrong.
Number: 41
Message: "An ISODE function unexpectedly failed.\nDiagnostics:%s"
Class: SI_Interface_Error

12.2.4 Errors associated with ADPU's:

Error: DBV_PDU_DECODEFAILED
Description: The snacc code could not decode the incoming PDU.
Number: 60
Message: "The received PDU could not be decoded"
Class: SI_Interface_Error

Error: DBV_PDU_ENCODEFAILED
Description: The snacc code could not encode the outgoing PDU.
Number: 61
Message: "An error occured while encoding the PDU"
Class: SI_Interface_Error

Error: DBV_PDU_NULLFIELD
Description: A compulsory field has not been given (could be a compulsory union member or first node in a list)
Number: 64
Message: "Missing compulsory field %s"
Class: SI_Interface_Error

Error: DBV_PDU_INVALIDBITSTRING
Description: A bit string has been filled incorrectly

Number: 65
Message: "The bit string %s is invalid\nValue:%s"
Class: SI_Interface_Error

Error: DBV_PDU_OUTOFRANGE
Description: An integer (usually an enumerated type) is out of its valid range
Number: 66
Message: "%s contains a value which is out of its range"
Class: SI_Interface_Error

Error: DBV_PDU_INVALIDOPTIONS
Description: An InitRequest options string contains options settings not permitted by the API (allowed options are specified by the #define ALLOWED_OPTIONS in the header file **dbvproto.h.**
Number: 67
Message: "The options string %s contains options not permitted by the API (see dbvproto.h ALLOWED_OPTIONS for allowed options)\nValue:%s"
Class: SI_Interface_Error

Error: DBV_PDU_NULLCHOICE
Description: Chosen choice not given
Number: 71
Message: "Field %s must be given"
Class: SI_Interface_Error

Error: DBV_PDU_INVALIDBOOL
Description: Invalid DbvBool value given
Number: 72
Message: "Invalid boolean value %s given"
Class: SI_Interface_Error

Error: DBV_PDU_INVALIDOID
Description: Invalid OID (Object Identifier) value given
Number: 73
Message: "Invalid oid value %s given"
Class: SI_Interface_Error

Error: DBV_PDU_EXTERNALOID
Description: Invalid oid (ObjectIdentifier) for a particular external
Number: 74
Message: "Invalid oid value %s for external %s"
Class: SI_Interface_Error

Error: DBV_PDU_OUTOFMEM
Description: Out of memory - internal alloc attempt returns NULL. May cause the program to crash.
Number: 75
Message: "Out of Memory"
Class: SI_Interface_Error

12.2.5 State Machine Errors

Error: DBV_SM_VIOLATION

Description: An event triggered by the service user or the remote origin/target association has violated the internal state machine.
Number: 80
Message: "Event %s is invalid for state %s"
Class: SI_Interface_error

Error: DBV_SM_ERROR
Description: An incoming event from the remote origin/target has violated the state machine. The current state will now be the error state.
Number: 81
Message: "Incoming event %s is an error for state %s"
Class: SI_Interface_Error

2.2.6 Specifying Target by Name

Error: DBV_FILE_FORMAT
Description: The format of the file being read is wrong.
Number: 100
Message: "File format error: Expected %s, read %s"
Class: SI_Interface_Error

Error: DBV_FILE_VALUE
Description: The file contains an invalid value
Number: 101
Message: "The value %s in the file is not valid"
Class: SI_Interface_Error

Error: DBV_NAME_ENTRY
Description: The specified name was not in the file
Number: 102
Message: "The entry %s is not in the name file"
Class: SI_Interface_Error

Error: DBV_FILE_ERROR
Description: An error was detected in the file
Number: 103
Message: "There is an error in the file"
Class: SI_Interface_Error

12.2.7 Miscellaneous Errors

Error: DBV_FILE
Description: A file could not be opened or does not exist.
Number: 110
Message: "The file %s could not be opened"
Class: SI_Interface_Error

Error: DBV_INTERNAL
Description: An unexpected error has occurred somewhere in the API
Number: 111
Message: "An internal error has occurred.\nDiagnostic:%s"
Class: SI_OperatingSystem_Error

12.2.8 Memory Errors

Error: DBV_MALLOC
Description: A request for memory failed. Not much that can be done here, except exit gracefully.
Number: 120
Message: "A request for memory failed: %s"
Class: SI_OperatingSystemError

12.2.9 ACSE Configuration File Errors

Error: DBV_ACSE_INVALID_ACN_VALUE
Description: An application context name is incorrect
Number: 130
Message: "The Application Context %s is invalid"
Class: SI_Interface_Error

Error: DBV_ACSE_FORMATERROR
Description: There is an error in the ACSE Configuration File
Number: 131
Message: "The format of the configuration file is incorrect\nDiagnostic:%s"
Class: SI_Interface_Error

Error: DBV_ACSE_MISSING_PCDL_VALUE
Description: The Presentation Context Definition List is missing
Number: 132
Message: "The Presentation Context Definition List is missing"
Class: SI_Interface_Error

13 SPECIFYING TARGETS BY NAME

13.1 Introduction

The DBV-OSI II API provides an easy way to connect to a target system by providing only a name (instead of a full address structure) in the `psTargetIdentification` parameter of the `DbvAssociateRequest()` function. This section describes the process which makes this possible.

When a target name is specified, the `DbvAssociateRequest()` function first looks in the **target_names.tbl** file. If the entry is not in this file, or the file is missing, then **isoentities** database is searched (only if ISODE has been installed). If the supplied name cannot be resolved by searching the files, the function will return an error (DBV_NAME_ENTRY) and fail.

13.2 The Name File Format

The procedure for specifying names is quite simple, but different for OSI and TCP addresses.

For TCP Addresses:

In the working directory in which the origin application will be residing a text file called **target_names.tbl** should be placed. The format of this file is as follows:

"Name", "Internet Address", "Socket Number";

For increased readability C-style comments (/* comment */) may be included in the file. Please note that multiple embedded comments are not allowed (/* comment 1 /* comment 2 */ comment 1 */ - is invalid).

For OSI Addresses:

For OSI addresses, the service-provider searches the ISODE file named **isoentites** for the entry supplied by the service-user. This file contains entries which allow the name to be resolved into a complete address.

The **isoentities** file will be located in the **SBIN** directory which was defined at ISODE compile time (maybe `/usr/etc` or `/usr/sbin`). The file `Make.defs` in the `config/` directory of the ISODE source tree should point to the correct directory.

The format for an entry in the **isoentities** database is as follows:

```
<host> <service> <aet> <paddr>
```

Each field is separated by a space, although double-quotes can be used to prevent separation.

An example entry is:

```
OSI-SIL     "SR/Z39.50 Service"     NULL \
      #321/Internet=sun001.sil.com
```

The `host` entry will be the name which must be subsequently used when completing the `psTargetIdentification->uNameOrAddress.pcTargetName` structure.

The `service` field MUST always be "SR/Z39.50 Service" (including quotes) as the API uses this along with the supplied host name to search the file.

The Application Entity Title (`aet`) may be left as NULL.

The `paddr` entry must contain the presentation address of the target system. It should include all of the communication methods common to both the origin and target systems, e.g.

```
#1050/Internet=amachine.acompany.acountry|Int-X25(80)=nnnnnnnnn|Janet=nnnnnn
```

Each type of address is separated by the bar (|) symbol.

13.2.1 Example target_names.tbl File

Below is a short example target name file:

```
/* target_names.tbl */
/* This file contains name to address mappings for the AssociateRequest()
function */

/* Satellites International Ltd */
"TCP-SIL", "193.195.22.1","2020";

/* End of name file */
```

Once this file and the **isoentities** file are in place they will be used by the `AssociateRequest()` function when it is called with the `DbvTarget->uNameOrAddress->pcTargetName` parameter set to a name. The files will be opened and searched for the appropriate entry. When the entry is found the address information is used to attempt to connect to the target system. If the entry specified by `pcTargetName` does not exist in the files, the DbvError structure will be completed and the function will return *`SI_NOTOK`*.

In the example above, for instance, the `eTargetNameOrAddress` field of the `psTarget` structure could be set to *`Target_Name`* and the `psTarget->uNameOrAddress->pcTargetName` field set to *`"TCP-SIL"`* before sending this structure as an argument to the `AssociateRequest()` function. The function will then automatically retrieve the information from the names file and attempt a connection.

Please note that the names file will be searched from the top down, so if duplicate entries exist the first occurrence will be read (unless commented out).

An example file will be included with the source code distribution.

14 THE ACSE CONFIGURATION FILE

14.1 Introduction

The ACSE Configuration file is used at both the origin and target. It is used to hold OSI specific configuration details which must be specified by the service user.

The origin ACSE Configuration file is used by the `AssociateRequest()` function to determine the ACSE parameters which will be present in the AAssociate.Request service primitive when initiating an association establishment.

The target ACSE Configuration file is used by the API to determine which application contexts and abstract syntaxes are supported by the application above the API. This allows the `DbvReceiveAssociateRequest()` function to determine the correct value for the **`peApplicationProtocol`** parameter and allows the `DbvAssociateResponse()` function to reject or accept the presentation contexts specified in the AAssociate.Request service primitive.

The ACSE configuration files for the origin and target have the same format which means that, as long as your origin requires and target accepts the same contexts, there is no reason why they cannot be pointed to the same file.

14.2 ACSE Configuration File Format

The file will provide the following ACSE parameters:

Application Context Name	The application context for the association. Permitted Values: "Basic_SR_Application_Context", "Basic_Z3950_Application_Context" Default Value: "Basic_SR_Application_Context" Note: The application on top of the API may support both protocols. In this case both of the above values shall be listed in the file.
Presentation Context Definition List	A list of abstract syntaxes supported by the application on top of the API. Permitted Values: Object Identifiers of the Record Syntaxes listed in Annex A4 of the document **D-008 API Definition** (e.g. "1.0.10163.5.1"). Default Value: Empty list

Note that the object identifiers for SR and Z39.50 are automatically added to the PCDL if present in the ACN.

The filename to be read is specified by the API function `SISetACSEFile()` (see section 9.2). If a new filename is not specified, the default is used. The default file must reside in the working directory of the origin application and be named **ACSE_configuration_file.tbl**.

The format for the file is detailed below:

```
Application_Context_Name, [Permitted Application Context Name Value];
Presentation_Context_Definition_List, [Permitted PCDL Value, ....., ..... ];
```

14.3 Example file

Below is a complete ACSE configuration file which supports both the SR and Z39.50 services:

```
/* Specimen ACSE Configuration File */
Application_Context_Name, "Basic_Z3950_Application_Context",
      "Basic_SR_Application_Context";

Presentation_Context_Definition_List, "1.2.840.10003.5.1",
      "1.2.840.10003.5.101","1.2.840.10003.5.102","1.2.840.10003.5.103"
      "1.2.840.10003.5.106", "1.0.10163.5.101","1.0.10163.5.102",
      "1.0.10163.5.103", "1.0.10163.5.106";
```

For each case, if the default value is needed, just the record header followed by a semi-colon is quite valid, e.g.

```
      Application_Context_Name;
```

This will set the application context name in the AAsociate.Request service primitive to its default value, or if used at the target set the protocol(s) understood by the application to the default (in this case `"Basic_SR_Application_Context"`).

For increased readability C-style comments (/* comment */) may be included in the file. Please note that multiple embedded comments are not allowed (/* comment 1 /* comment 2 */ comment 1 */ - is invalid).

The API expects each record to appear in the order listed above. Changing the order will confude the API.

14.4 Further Information

Further information on the ACSE configuration file is contained in D-008 API Definition, Section 3.1.1.1 and Section 4.4.

15. REGISTERING AN OSI APPLICATION USING ISODE

15.1 Introduction

The ISO Development Envirmonment (ISODE) has two daemon processes for starting applications, `iaed` and `tsapd`. The ISODE documentation recommends the use of the `iaed`.

Before an application can be contacted via an association, the target application needs to be registered.

This section assumes that system users are running a window manager and use window processes such as `xterms` or `cmdtools`.

Remember that the daemon process may be a root process. This means that target processes will start as root owned also. Care must be taken to ensure that unauthorised root access to a system is not gained!

15.3 Using iaed

`iaed` uses the ISODE implementation of the X.500 directory services (QUIPU) to gain information on which application services are supported by the local system.

When the `iaed` is started it queries the QUIPU daemon (`ros.quipu`) and loads in the local application services. These services are obtained from the X.500 directory database. This means that all application services must be entered into the database before they can be used.

15.3.1 Registering An Application

The following example is a modification of the example in **ISODE Volume 2 Administrator's Guide: Installation - Chapter 4.4 Registering OSI Application Services**. It will create a new application entity which will start up the DBV-OSI II test program on the display specified. This is ideal for development purposes.

It is assumed here that a DSA is already present (see **ISODE Volume 5 Administrators Guide: Directory Services -** The whole document) and that the ISODE directories are in the system search path.

1. Create a file containing the following (the filename is not important, maybe `new_service`):

```
objectClass= top & applicationEntity & quipuObject & iSODEApplicationEntity
cn= xterm001-test
presentationAddress= #1024/Internet=sun001.sil.com
acl=
```

```
execVector= /usr/openwin/bin/xterm -display xterm001:0.0 -e
/usr/dbvosi/release/libtest/apitesttargetslave OSI
```

Change the `cn=` entry to the name of your choice. The name is for reference in the directory and can be specified as anything you wish.

The `presentationAddress=` entry should be set to a unique (unused) number (between 1024 and 2047 for local services) and the OSI communities the local system supports.

The `execVector=` entry should point to the application to be executed. The entry should be on one continuous line, i.e. contain only a return character at the end of the line. The first parameter being the application and the rest the parameters to pass to it.

The `objectClass=` entry should not be changed.

A file containing the following entry might be more suitable for a finalised, on-line OSI service:

```
objectClass= top & applicationEntity & quipuObject & iSODEApplicationEntity
cn= SR/Z39.50 Service
presentationAddress= #1024/Internet=amachine.acompany.acountry|Int-
X25(80)=nnnnnnnn
acl=
execVector= /usr/dbvosi/release/ositarget OSI
```

2. Start `dish` in the background, bind as the manager and move to the location in the DIT where the services are to be registered:

```
% setenv DISHPROC "127.0.0.1 `expr $$ + 32768`"
% bind -u <<DN of DSA Manager>>
   e.g. bind -u "@c=GB@o=Satellites International Ltd@ou=Software@cn=dsaManager"
% moveto << Application Services Directory >>
   e.g. moveto "@c=GB@o=Satellites International Ltd@ou=Software@cn=Application
Services"
```

3. Add the new service, e.g.:

```
% add "cn=xterm001-test" ./new_service -noedit
```

If the new service has been added correctly, then a message similar to

```
Added @c=GB@o=Satellites International Ltd@ou=Software@cn=Application Services@cn=xterm001-test
```

will appear on the screen.

A service may also be removed using the remove command, e.g.:

```
remove "@c=GB@o=Satellites International Ltd@ou=Software@cn=Application Services@cn=xterm001-test"
```

A session should be terminated using the command `unbind`.

After adding and removing application services `iaed` must be restarted. this can be done using the command

```
kill -HUP <<pid>>
```

The Satellites International directory service is used above as an example. Obviously, the local system will have its own, different structure which must be used.

15.3.2 Starting iaed

When using directory services, `iaed` must be running. This is done by typing:

```
iaed -D "@c=GB@o=Satellites International Ltd@ou=Software@cn=Application Services"
```

and substituting the example directory for the one where services are registered on the local system.

`iaed` can also be started at system boot-time. For details on how to do this refer to **ISODE Volume 2: Administrator's Guide: Installation - Chapter 4.4.2**.

15.3 Using tsapd

Start `tsapd` by moving to the directory in which it is located (or place that directory in your search path) and type `tsapd >& /dev/null` at the command line. Depending on the configuration of the OSI system, it might be necessary to be the root user to do this. The daemon may also be initiated at system startup. Refer to **ISODE Volume 2 Administrator's Guide Installation - Chapter 4** for details on how to do this.

`tsapd` can also be started in 'debug mode' by not redirecting the output to `/dev/null`. This will print connection information to the screen, but the daemon will terminate after every association.

When the `tsapd` receives a `T-CONNECT.INDICATION` the **isoservices** file is read to see if the requested presentation address has a local application associated with it. If it does, then the daemon starts the application. This means that a target DBV-OSI II application must have an entry in the **isoservices** file.

15.3.1 isoservices Examples

For local development purposes it is usually useful for the target executable to attach itself to a display. This can be done with an entry similar to the following:

```
"tsap/SR-Z39.50"  #1024 /usr/openwin/bin/xterm -display xterm01:0.0 -e
/usr2/dbvosi/src/api/apitesttargetslave OSI
```

Then place an entry in the **isoentities** file, such as:

```
xterm01-test      "SR/Z39.50 Service"     NULL  #1024/Internet=localhost
```

Now by specifying `psTarget->uNameOrAddress->pcTargetName` as `"xterm01-test"` and calling `DbvAssociateRequest()` an association will be attempted. When successful the api test program would be started inside an `xterm` process on the specified display. This process may also be easily debugged using `gdb`.

This method has the advantage that many developers can test on one system but have the target programs directed to their own display.

When a system is to go on-line an **isoservices** entry such as the following may be more appropriate:

```
"tsap/SR-Z39.50"  #1024 /usr/dbvosi/target/dbvtarget OSI
```

Please note that:

- Local services should have presentation addresses between #1024 and #2047.
- All entries in the **isoservices** file should be on one line (no return characters) to work correctly!
- All entries should be placed in the **locally defined services** section of the file.

16 CONFIGURING AN INTERNET DAEMON

16.1 Using targetd

The supplied `targetd` program can be useful when developing as it can be told to listen on any port number and execute any program. This is helpful if different developers are using the same system. The program can be found in the `release/targetd/` directory of the `dbvosi` tree.

To start the program on the default port and default executable, move to the correct directory and type `targetd`.

The program may also be configured to listen on a different port number, or executable. Here are some examples:

Configure the target program to listen on TCP/IP port 2060:

```
targetd –p 2060
```

Configure the targetd program to listen on TCP/IP port 2050 and point to a different executable:

```
targetd –p 2050 –e /usr/dbvosi/development/target/target–test
```

16.1 Using inetd

When a completed TCP/IP SR/Z39.50 system goes on-line, it might be desirable to add the service to the system `inetd` daemon. This is quite a simple process .....

<< desired port no. >> << path/executable>> <<TCP>>

Annex A: Data Definitions

A.1 Constants and Type Definitions

```
/* Return values of the API functions: */
#define SI_OK                              0
#define SI_NOTOK                           1

/*Call modes for the API functions: */
#define CALL_BLOCKING                     -1L
#define CALL_ASYNCHRONOUS                  0L

/* Invalid association identifier: */
#define NO_ASSOCIATION_ID_ASSIGNED        -1

typedef long            DbvAssocId;
typedef long            DbvSecs;
typedef long            DbvErrorCode;
typedef long            DbvAbortReason;
typedef unsigned long   DbvFds;
typedef unsigned long   DbvPortNumber;
```

A.2 Enumeration Types

```
/* Responses for releasing an association */
typedef enum DbvReleaseRspReason {
      Normal = 0,
      Not_Finished = 1
} DbvReleaseRspReason;

typedef enum DbvReleaseRspResult {
      Affirmative = 0,
      Negative = 1
} DbvReleaseRspResult;

typedef struct DbvReleaseRsp {
      DbvReleaseRspReason      *peReleaseRspReason;
      DbvReleaseRspResult       eReleaseRspResult;
} DbvReleaseRsp;

/* Enumeration Types */

/* Events requiring a read operation:
typedef enum EventLocation
            API_No_Event (0),
            API_User_Event (1),
            API_Provider_Event (2)
} EventLocation;
typedef enum EventLocation {
            API_No_Event =0,
            API_User_Event =1,
```

```
            API_Provider_Event =2
} EventLocation;*/

/* New definition of DbvEventLocation */
typedef struct DbvEventLocation {
      DbvFds      **ppulUserEvent;
      DbvAssocId  **pplProviderEvent;
} DbvEventLocation;


/* Protocols supported: */
typedef enum DbvApplicationProtocol {
            Unknown_Protocol = 0,
            Unidentified_Protocol = 1,
            SR_Protocol = 2,
            Z3950_Protocol = 3
} DbvApplicationProtocol;

/* Protocol stacks supported: */
typedef enum DbvProtocolStack {
            Unknown_Stack = 0,
            OSI_Stack = 1,
            TCP_Stack = 2
} DbvProtocolStack;

/* Associate Response: */
typedef enum DbvAssociateRsp {
            Accepted = 0,
            Rejected_Perm = 1,
            Rejected_Tran = 2
} DbvAssociateRsp;

/* Abstract syntaxes: */
typedef enum DbvAbstractSyntax {
            No_Syntaxes_Provided = 0,
            No_Common_Syntax = 1,
            Common_Syntaxes = 2
} DbvAbstractSyntax;

/* Association states during association/connection establishment phase: */
typedef enum DbvAssociateState {
            Associate_Null  = 0,
            Associate_Accepted  = 1,
            Associate_Rejected  = 2,
            Associate_Pending  = 3
} DbvAssociateState;

/* Associate rejection: */
typedef enum DbvAssociateRejectSource {
            ACSE_User = 0,
            ACSE_Provider = 1
} DbvAssociateRejectSource;


typedef enum DbvAssociateRejectResult {
            Rejected_Permanent = 0,
            Rejected_Transient = 1
} DbvAssociateRejectResult;
```

```
typedef enum DbvAssociateRejectReason {
            No_Reason_Given = 0,
            No_Common_ACSE_Version = 1,
            Application_Context_Not_Supported = 2
} DbvAssociateRejectReason;

/* Association states during association/connection release phase: */
typedef enum DbvReleaseState {
            Release_Null  = 0,
            Release_Accepted  = 1,
            Release_Rejected  = 2,
            Release_Pending  = 3
} DbvReleaseState;

typedef enum DbvReleaseReq {
            ReleaseReq_Normal = 0,
            ReleaseReq_Urgent = 1,
            ReleaseReq_Undefined = 2
} DbvReleaseReq;

/* Association state upon return of an API function: */
typedef enum DbvAssociationState {
            Association_Open = 0,
            Association_Closed = 1,
            Association_Unknown_State = 2
} DbvAssociationState;

/* Error classes: */
typedef enum DbvErrorClass {
            SI_Communication_Error = 0,
            SI_Interface_Error = 1,
            SI_OperatingSystem_Error = 2,
            SI_Abort = 3
} DbvErrorClass;



/* Abort sources: */
typedef enum DbvAbortSource {
            SI_ACSE_User = 0,
            SI_ACSE_Provider = 1,
            SI_Presentation_User = 2,
            SI_Presentation_Provider = 3,
            SI_Transport_Provider = 4
} DbvAbortSource;

/* Permitted SR-/Z39.50 events at the Origin system */
typedef enum DbvOriginEvent {
            OR_InitializeConf = 0,
            OR_SearchConf = 1,
            OR_PresentConf = 2,
            OR_DeleteResultSetConf = 3,
            OR_ScanConf = 4,
            OR_ResourceReportConf = 5,
            OR_SegmentationInd = 6,
            OR_SortConf = 7,
            OR_ExtendedServicesConf = 8,
            OR_CloseConf = 9,
            OR_CloseInd = 10,
```

```
/*          OR_ReleaseInd = 11,  Target is not allowed to close the
                                        association */
            OR_Invalid_APDU = 11
} DbvOriginEvent;

/* Permitted SR-/Z39.50 events at the Target system */
typedef enum DbvTargetEvent {
            TA_InitializeInd = 0,
            TA_SearchInd = 1,
            TA_PresentInd = 2,
            TA_DeleteResultSetInd = 3,
            TA_ScanInd = 4,
            TA_ResourceReportInd = 5,
            TA_SortInd = 6,
            TA_ExtendedServicesInd = 7,
            TA_CloseInd = 8,
            TA_CloseConf = 9,
            TA_ReleaseInd = 10,
            TA_Invalid_APDU = 11
} DbvTargetEvent;

/* Target identification */
typedef enum DbvTargetNameOrAddress {
            Target_Name = 0,
            Target_Address = 1
} DbvTargetNameOrAddress;

/* Address types */
typedef enum DbvOsiOrInternetAddress {
            Osi_Address = 0,
            Internet_Address = 1
} DbvOsiOrInternetAddress;


typedef enum DbvTraceLevel {
      Trace_Off = 0,
      Trace_Low = 1,
      Trace_High = 2
} DbvTraceLevel;
```

A.3 Structure/Union Types

A.3.1 Structures associated with Association Handling functions

```
typedef struct DbvAssociateReject {
      DbvAssociateRejectSource      *peAssociateRejectSource;
      DbvAssociateRejectResult      *peAssociateRejectResult;
      DbvAssociateRejectReason      *peAssociateRejectReason;
} DbvAssociateReject;

typedef struct DbvProtocolAndSyntaxes {
      DbvApplicationProtocol          *peApplicationProtocol;
```

```
      DbvAbstractSyntax         *peAbstractSyntax;
      char                    **ppcCommonSyntaxes;
/* valid <--> peAbstractSyntax == Common_Syntaxes */
} DbvProtocolAndSyntaxes;

typedef union DbvAssociateResult {
      DbvProtocolAndSyntaxes        *psProtocolAndSyntaxes;
      DbvAssociateReject            *psAssociateReject;
} DbvAssociateResult;


typedef struct DbvOsiAddress {
      char        *pcPsel;
      char        *pcSsel;
      char        *pcTsel;
      char        *pcNsap;
} DbvOsiAddress;

typedef struct DbvInternetAddress {
      char        *pcHostAddress;
      DbvPortNumber    ulPortNumber;
} DbvInternetAddress;

typedef struct DbvTargetInformation {
      DbvApplicationProtocol              *peTargetSupportedProtocol;
      DbvOsiOrInternetAddress               eOsiOrInternetAddress;
      union {
            DbvOsiAddress           *psOsiAddress;
            DbvInternetAddress      *psInternetAddress;
      }                               uOsiOrInternetAddress;
} DbvTargetInformation;


typedef struct DbvTarget {
      DbvTargetNameOrAddress              eTargetNameOrAddress;
      union {
            char              *pcTargetName;
            DbvTargetInformation    *psTargetInformation;
      }                             uNameOrAddress;
} DbvTarget;
```

A.3.2 Structures associated with SR-/Z39.50 functions

```
typedef struct DbvOriginData {
  DbvOriginEvent        eOriginEvent;
  union {
    DbvInitializeRsp        *psInitializeRsp;
    DbvSearchRsp      *psSearchRsp;
    DbvPresentRsp     *psPresentRsp;
    DbvDeleteResultSetRsp   *psDeleteResultSetRsp;
    DbvScanRsp              *psScanRsp;
    DbvResourceReportRsp    *psResourceReportRsp;
    DbvSegmentReq     *psSegmentReq;
    DbvSortRsp              *psSortRsp;
    DbvCloseReq               *psCloseReq;
    DbvExtendedServicesRsp  *psExtendedServicesRsp;
```

```
/*     DbvReleaseReq        *peReleaseReq;   Target is not allowed to close the
                                             association */
    DbvOctetString          *psInvalidAPDU;
  }   uOriginData;
} DbvOriginData;

typedef struct DbvTargetData {
  DbvTargetEvent  eTargetEvent;
  union {
    DbvInitializeReq          *psInitializeReq;
    DbvSearchReq        *psSearchReq;
    DbvPresentReq       *psPresentReq;
    DbvDeleteResultSetReq     *psDeleteResultSetReq;
    DbvScanReq                *psScanReq;
    DbvResourceReportReq      *psResourceReportReq;
    DbvSortReq                *psSortReq;
    DbvCloseReq               *psCloseReq;
    DbvExtendedServicesReq          *psExtendedServicesReq;
    DbvReleaseReq       *peReleaseReq;
    DbvOctetString              *psInvalidAPDU;
  }               uTargetData;
} DbvTargetData;
```

A.3.3 Structures common to all functions

```
typedef struct DbvAbort {
  DbvAbortSource          eAbortSource;
  DbvAbortReason          lAbortReason;
  char                         *pcDiagnostics;
} DbvAbort;

typedef struct DbvError {
  DbvErrorClass         eErrorClass;
  union {
    DbvErrorCode    lIfOsCoError;
    DbvAbort                *psAbort;
  }               uErrorCode;
  DbvAssociationState   eAssociationState;
} DbvError;
```

Annex B: SR-/Z39.50 Data Structures

B.1 Global type and structure definitions

```
/* ------------------------------------------------------------*/
/*               Types and Sub-Structures                      */
/* ------------------------------------------------------------*/

typedef unsigned long   Ulong;
typedef char*           DbvProtocolVersion;
typedef char*           DbvOptions;


typedef enum DbvBool {
             DBV_FALSE = 0,
             DBV_TRUE  = 1
} DbvBool;


typedef char*           DbvObjectIdentifier;



/* -------------   OctetString   ----------------------------- */

typedef struct DbvOctetString {
Ulong      ulLength;
char      *pcValue;
} DbvOctetString;



/* -------------   DbvVisibleString   ----------------------------- */

typedef char*          DbvVisibleString;


/* -------------   DbvGeneralizedTime   ----------------------------- */

typedef char*          DbvGeneralizedTime;


/* -------------   DbvInternationalString   ----------------------------- */

typedef struct DbvInternationalString {
Ulong      ulLength;
char      *pcValue;
} DbvInternationalString;



/* -------------   DbvGeneralString   ----------------------------- */
```

```
typedef struct DbvGeneralString {
Ulong      ulLength;
char      *pcValue;
} DbvGeneralString;



/* --------------  ASN1External  ----------------------- */

typedef enum DbvExternalValueType {
                     EVT_SingleASN1Type = 1,
                     EVT_OctetAligned   = 2,
                     EVT_Arbitrary      = 3
} DbvExternalValueType;

typedef enum DbvASN1Type {
              AT_UnstructuredText      = 1,
              AT_OpacRecord            = 2,
              AT_SummaryRecord         = 3,
              AT_ResourceReport1       = 4,
              AT_ResourceReport2       = 5,
              AT_EsTaskPackage         = 6,
              AT_UpdateExtendedService = 7,
              AT_UnknownType           = 8
} DbvASN1Type;


typedef struct DbvSingleASN1Type {
DbvASN1Type          eASN1Type;
union {
    struct DbvInternationalString   *psUnstructuredText;
    struct DbvOpacRecord            *psOpacRecord;
    struct DbvSummaryRecord         *psSummaryRecord;
    struct DbvResourceReport1       *psResourceReport1;
    struct DbvResourceReport2       *psResourceReport2;
    struct DbvEsTaskPackage         *psEsTaskPackage;
    struct DbvUpdateExtendedService *psUpdateExtendedService;
    struct DbvOctetString           *psUnknownType;
  }               uASN1;
} DbvSingleASN1Type;


typedef struct DbvASN1External {
DbvObjectIdentifier      pcObjectIdentifier;
DbvExternalValueType     eValueType;
union {
    DbvSingleASN1Type      *psSingleASN1Type;
    DbvOctetString         *psOctetAligned;
    DbvOctetString         *psArbitrary;
  }                   uExternalValue;
} DbvASN1External;



/* --------------  Other Information  ----------------------- */

typedef enum DbvInformationType {
                   IT_CharacterInfo    = 1,
                   IT_BinaryInfo       = 2,
```

```
                  IT_ExternalInfo     = 3,
                  IT_ObjectIdentifier = 4
} DbvInformationType;


typedef struct DbvInfoCategory {
DbvObjectIdentifier    pcCategoryTypeId;
Ulong                  ulCategoryValue;
} DbvInfoCategory;


typedef struct DbvOtherInformation {
DbvInfoCategory       *psInfoCategory;
DbvInformationType     eInformationType;
union {
    DbvInternationalString      *psCharacterInfo;
    DbvOctetString              *psBinaryInfo;
    DbvASN1External             *psExternallyDefinedInfo;
    DbvObjectIdentifier          pcObjectIdentifier;
  }                uInformation;
} DbvOtherInformation;
```

B.2 APDU Datastructures

```
/* ------------------    INITIALIZE   ---------------------------------- */


typedef struct DbvInitializeReq {
DbvOctetString         *psReferenceId;
DbvProtocolVersion      pcProtocolVersion;
DbvOptions              pcOptions;
Ulong                   ulPreferredMessageSize;
Ulong                   ulMaximumMessageSize;
DbvAuthentication      *psAuthentication;
DbvInternationalString *psImplementationId;
DbvInternationalString *psImplementationName;
DbvInternationalString *psImplementationVersion;
DbvASN1External        *psUserInformationField;
DbvOtherInformation   **ppsOtherInformation;
} DbvInitializeReq;



typedef struct DbvInitializeRsp {
DbvOctetString         *psReferenceId;
DbvProtocolVersion      pcProtocolVersion;
DbvOptions              pcOptions;
Ulong                   ulPreferredMessageSize;
Ulong                   ulMaximumMessageSize;
DbvBool                 fResult;
DbvInternationalString *psImplementationId;
DbvInternationalString *psImplementationName;
DbvInternationalString *psImplementationVersion;
DbvASN1External        *psUserInformationField;
DbvOtherInformation   **ppsOtherInformation;
```

```
} DbvInitializeRsp;


/* ------------------    SEARCH   ---------------------------------- */


typedef enum DbvResultSetStatus {
                    RSS_Subset  = 1,
                    RSS_Interim = 2,
                    RSS_None    = 3,
                    RSS_Invalid = -1
} DbvResultSetStatus;


typedef enum DbvPresentStatus {
                  PS_Success   = 0,
                  PS_Partial_1 = 1,
                  PS_Partial_2 = 2,
                  PS_Partial_3 = 3,
                  PS_Partial_4 = 4,
                  PS_Failure   = 5,
                  PS_Invalid   = -1
} DbvPresentStatus;



typedef struct DbvSearchReq {
DbvOctetString           *psReferenceId;
Ulong                     ulSmallSetUpperBound;
Ulong                     ulLargeSetLowerBound;
Ulong                     ulMediumSetPresentNumber;
DbvBool                   fReplaceIndicator;
DbvInternationalString    sResultSetName;
DbvInternationalString  **ppsDatabaseNames;
DbvElementSetNames       *psSmallSetElementNames;
DbvElementSetNames       *psMediumSetElementNames;
DbvObjectIdentifier       pcPreferredRecordSyntax;
DbvQuery                 *psQuery;
DbvOtherInformation     **ppsAdditionalSearchInfo;
DbvOtherInformation     **ppsOtherInformation;
} DbvSearchReq;



typedef struct DbvSearchRsp {
DbvOctetString         *psReferenceId;
Ulong                   ulResultCount;
Ulong                   ulNumberOfRecordsReturned;
Ulong                   ulNextResultSetPosition;
DbvBool                 fSearchStatus;
DbvResultSetStatus     *peResultSetStatus;
DbvPresentStatus       *pePresentStatus;
DbvRecords             *psRecords;
DbvOtherInformation   **ppsAdditionalSearchInfo;
DbvOtherInformation   **ppsOtherInformation;
} DbvSearchRsp;
```

```
/* ------------------    PRESENT   ---------------------------------- */


typedef struct DbvPresentReq{
DbvOctetString             *psReferenceId;
DbvInternationalString      sResultSetId;
Ulong                       ulResultSetStartPoint;
Ulong                       ulNumberOfRecordsRequested;
DbvRange                   **ppsAdditionalRanges;
DbvRecordComposition        *psRecordComposition;
DbvObjectIdentifier         pcPreferredRecordSyntax;
Ulong                       *pulMaxSegmentCount;
Ulong                       *pulMaxRecordSize;
Ulong                       *pulMaxSegmentSize;
DbvOtherInformation        **ppsOtherInformation;
} DbvPresentReq;


typedef struct DbvPresentRsp {
DbvOctetString         *psReferenceId;
Ulong                   ulNumberOfRecordsReturned;
Ulong                   ulNextResultSetPosition;
DbvPresentStatus        ePresentStatus;
DbvRecords             *psRecords;
DbvOtherInformation   **ppsOtherInformation;
} DbvPresentRsp;


/* ------------------    DELETE-RESULT-SET   ------------------------------ */


typedef enum DbvDeleteFunction {
                  DF_List     = 0,
                  DF_All      = 1,
                  DF_Invalid  = -1
} DbvDeleteFunction;


typedef struct DbvDeleteResultSetReq {
DbvOctetString               *psReferenceId;
DbvDeleteFunction             eDeleteFunction;
DbvInternationalString      **ppsResultSetId;
DbvOtherInformation         **ppsOtherInformation;
} DbvDeleteResultSetReq;


typedef struct DbvDeleteResultSetRsp {
DbvOctetString             *psReferenceId;
DbvDeleteSetStatus          eDeleteOperationStatus;
DbvListStatus             **ppsDeleteListStatuses;
Ulong                      *pulNumberNotDeleted;
DbvListStatus             **ppsBulkStatuses;
```

```
DbvInternationalString     *psDeleteMessage;
DbvOtherInformation       **ppsOtherInformation;
} DbvDeleteResultSetRsp;


/* ------------------    SEGMENT   ------------------------------ */


typedef struct DbvSegmentReq {
DbvOctetString         *psReferenceId;
Ulong                   ulNumberOfRecordsReturned;
DbvNamePlusRecord     **ppsSegmentRecords;
DbvOtherInformation   **ppsOtherInformation;
} DbvSegmentReq;


/* ------------------    RESOURCE-REPORT   ---------------------------- */


typedef enum DbvResourceReportStatus {
                        RRS_Success   = 0,
                        RRS_Partial   = 1,
                        RRS_Failure_1 = 2,
                        RRS_Failure_2 = 3,
                        RRS_Failure_3 = 4,
                        RRS_Failure_4 = 5,
                        RRS_Failure_5 = 6,
                        RRS_Failure_6 = 7,
                        RRS_Invalid   = -1
} DbvResourceReportStatus;



typedef struct DbvResourceReportReq {
DbvOctetString            *psReferenceId;
DbvOctetString            *psOperationId;
DbvObjectIdentifier        pcPrefResourceReportFormat;
DbvOtherInformation      **ppsOtherInformation;
} DbvResourceReportReq;


typedef struct DbvResourceReportRsp {
DbvOctetString             *psReferenceId;
DbvResourceReportStatus     eResourceReportStatus;
DbvASN1External            *psResourceReport;
DbvOtherInformation       **ppsOtherInformation;
} DbvResourceReportRsp;


/* ------------------    SCAN   ------------------------------ */


typedef enum DbvScanStatus {
              SCS_Success   = 0,
              SCS_Partial_1 = 1,
```

```
              SCS_Partial_2 = 2,
              SCS_Partial_3 = 3,
              SCS_Partial_4 = 4,
              SCS_Partial_5 = 5,
              SCS_Failure   = 6,
              SCS_Invalid   = -1
} DbvScanStatus;



typedef struct DbvScanReq {
DbvOctetString              *psReferenceId;
DbvInternationalString     **ppsDatabaseNames;
DbvObjectIdentifier          pcAttributeSetId;
DbvAttributesPlusTerm        sAttributesPlusTerm;
Ulong                       *pulStepSize;
Ulong                        ulNumberOfTermsRequested;
Ulong                       *pulPreferredPositionInResponse;
DbvOtherInformation        **ppsOtherInformation;
} DbvScanReq;


typedef struct DbvScanRsp {
DbvOctetString         *psReferenceId;
Ulong                  *pulStepSize;
DbvScanStatus           eScanStatus;
Ulong                   ulNumberOfEntriesReturned;
Ulong                  *pulPositionOfTerm;
DbvListEntries         *psEntries;
DbvObjectIdentifier     pcAttributeSetId;
DbvOtherInformation   **ppsOtherInformation;
} DbvScanRsp;



/* ------------------    SORT   ------------------------------ */


typedef enum DbvSortStatus {
                       SOS_Success   = 0,
                       SOS_Partial_1 = 1,
                       SOS_Failure   = 2,
                       SOS_Invalid   = -1
} DbvSortStatus;


typedef enum DbvSortResultSetStatus {
                       SRSS_Empty     = 0,
                       SRSS_Interim   = 1,
                       SRSS_Unchanged = 2,
                       SRSS_None      = 3,
                       SRSS_Invalid   = -1
} DbvSortResultSetStatus;



typedef struct DbvSortReq {
DbvOctetString           *psReferenceId;
```

```
DbvInternationalString  **ppsInputResultSetName;
DbvInternationalString    sSortedResultSetName;
DbvSortKeySpec          **ppsSortSequence;
DbvOtherInformation     **ppsOtherInformation;
} DbvSortReq;


typedef struct DbvSortRsp {
DbvOctetString             *psReferenceId;
DbvSortStatus               eSortStatus;
DbvSortResultSetStatus     *peResultSetStatus;
DbvDiagRec                **ppsDiagnostics;
DbvOtherInformation       **ppsOtherInformation;
} DbvSortRsp;



/* ------------------    CLOSE   ----------------------------- */


typedef enum DbvCloseReason {
               CR_Finished          = 0,
               CR_ShutDown          = 1,
               CR_SystemProblem     = 2,
               CR_CostLimit         = 3,
               CR_Resources         = 4,
               CR_SecurityViolation = 5,
               CR_ProtocolError     = 6,
               CR_LackOfActivity    = 7,
               CR_PeerAbort         = 8,
               CR_Unspecified       = 9,
               CR_Invalid           = -1
} DbvCloseReason;




typedef struct DbvCloseReq {
DbvOctetString                    *psReferenceId;
DbvCloseReason                     eCloseReason;
DbvInternationalString            *psDiagnosticInformation;
DbvObjectIdentifier                pcResourceReportFormat;
DbvASN1External                   *psResourceReport;
DbvOtherInformation              **ppsOtherInformation;
} DbvCloseReq;




/* ------------------    EXTENDED-SERVICES   ----------------------------- */

typedef enum DbvExtServicesFunctions {
                   ESF_Create = 1,
                   ESF_Delete = 2,
                   ESF_Modify = 3,
                   ESF_Invalid      = -1
} DbvExtServicesFunctions;
```

```
typedef enum DbvExtServicesWaitAction {
                  ESWA_Wait              = 1,
                  ESWA_WaitIfPossible    = 2,
                  ESWA_DontWait          = 3,
                  ESWA_DontReturnPackage = 4,
                  ESWA_Invalid           = -1
} DbvExtServicesWaitAction;


typedef enum DbvExtServicesOperationStatus {
                  ESOS_Done    = 1,
                  ESOS_Queued  = 2,
                  ESOS_Failure = 3,
                  ESOS_Invalid = -1
} DbvExtServicesOperationStatus;



typedef struct DbvExtendedServicesReq {
DbvOctetString                       *psReferenceId;
DbvExtServicesFunctions               eFunction;
DbvObjectIdentifier                   pcPackageType;
DbvInternationalString               *psPackageName;
DbvInternationalString               *psUserId;
DbvIntUnit                           *psRetentionTime;
DbvPermissions                      **ppsPermissions;
DbvInternationalString               *psDescription;
DbvASN1External                      *psOriginSpecificParameter;
DbvExtServicesWaitAction              eWaitAction;
DbvInternationalString               *psElements;
DbvOtherInformation                 **ppsOtherInformation;
} DbvExtendedServicesReq;


typedef struct DbvExtendedServicesRsp {
DbvOctetString                       *psReferenceId;
DbvExtServicesOperationStatus         eOperationStatus;
DbvDiagRec                          **ppsDiagnostics;
DbvASN1External                      *psTaskPackage;
DbvOtherInformation                 **ppsOtherInformation;
} DbvExtendedServicesRsp;
```

B.3 APDU Substructures

```
/* --------------  Authentication ----------------------- */

typedef enum DbvAuthenticationType {
                 AT_Open       = 1,
                 AT_IdPassword = 2,
                 AT_Anonymous  = 3,
                 AT_Other      = 4
} DbvAuthenticationType;


typedef struct DbvIdPassword {
```

```
DbvInternationalString      *psGroupId;
DbvInternationalString      *psUserId;
DbvInternationalString      *psPassword;
} DbvIdPassword;


typedef struct DbvAuthentication{
DbvAuthenticationType   eAuthenticationType;
union {
    DbvVisibleString           psOpen;
    DbvIdPassword               *psIdPassword;
    DbvASN1External             *psOther;
  }                 uAuthentication;
} DbvAuthentication;



/* --------------  DbvElementSetNames  ----------------------- */

typedef enum DbvElementSetType {
                  EST_Generic          = 1,
                  EST_DatabaseSpecific = 2
} DbvElementSetType;


typedef struct DbvElementSetsDbSpecific {
DbvInternationalString       *psDatabaseName;
DbvInternationalString       *psElementSetName;
} DbvElementSetsDbSpecific;


typedef struct DbvElementSetNames {
DbvElementSetType      eElementSetType;
union {
    DbvInternationalString         *psGeneric;
    DbvElementSetsDbSpecific      **ppsElementSetsDbSpecific;
  }               uElementSet;
} DbvElementSetNames;



/* --------------  DbvQuery  ----------------------- */


typedef enum DbvQueryType {
             QT_Private         = 1,
             QT_Rpn             = 2,
             QT_Iso8777         = 3,
             QT_Z3958           = 4,
             QT_Erpn            = 5,
             QT_NaturalLanguage = 6
} DbvQueryType;


typedef  enum DbvRPNStructureType {
                  RST_RpnLopRopOperator = 1,
                  RST_RpnOperand        = 2
} DbvRPNStructureType;
```

```
typedef enum DbvOperandType {
                  OT_AttributePlusTerm       = 1,
                  OT_ResultSetId             = 2,
                  OT_ResultSetPlusAttributes = 3
} DbvOperandType;


typedef enum DbvTermType {
              TT_General          = 1,    /* the original, Version 2 type */
              TT_Numeric          = 2,
              TT_SimpleCharacter  = 3,
              TT_ObjectIdentifier = 4,
              TT_DateTime         = 5,
              TT_External         = 6,
              TT_IntegerAndUnit   = 7,
              TT_Null             = 8
} DbvTermType;


typedef enum DbvOperatorType {
                  OT_Or        = 1,
                  OT_And       = 2,
                  OT_AndNot    = 3,
                  OT_Proximity = 4
} DbvOperatorType;

typedef enum DbvProximityUnitType {
                     PUT_Known   = 1,
                     PUT_Private = 2
} DbvProximityUnitType;


typedef enum DbvProximityRelationType {
                         PRT_LessThan           = 1,
                         PRT_LessThanOrEqual    = 2,
                         PRT_Equal              = 3,
                         PRT_GreaterThanOrEqual = 4,
                         PRT_GreaterThan        = 5,
                         PRT_NotEqual           = 6,
                         PRT_Invalid            = -1
} DbvProximityRelationType;


typedef enum DbvKnownProximityUnit {
                       KPU_Character   = 1,
                       KPU_Word        = 2,
                       KPU_Sentence    = 3,
                       KPU_Paragraph   = 4,
                       KPU_Section     = 5,
                       KPU_Chapter     = 6,
                       KPU_Document    = 7,
                       KPU_Element     = 8,
                       KPU_SubElement  = 9,
                       KPU_ElementType = 10,
                       KPU_Byte        = 11,
                       KPU_Invalid     = -1
} DbvKnownProximityUnit;
```

```
typedef struct DbvProximityUnitCode {
DbvProximityUnitType    eProximityUnitType;
union {
    DbvKnownProximityUnit  eKnownProximityUnit;
    Ulong                  ulPrivate;
  }                  uProximityUnit;
} DbvProximityUnitCode;


typedef struct DbvProximityOperator {
DbvBool                   *pfExclusion;
Ulong                      ulDistance;
DbvBool                    fOrdered;
DbvProximityRelationType   eProximityRelationType;
DbvProximityUnitCode       sProximityUnitCode;
} DbvProximityOperator;


typedef struct DbvOperator {
DbvOperatorType       eOperatorType;
DbvProximityOperator *psProximityOperator; /* <--> eOperatorType == OT_Proximity
*/
} DbvOperator;


    /* ---- DbvAttributeValue ----( not the final definition ) ---- */

typedef enum DbvAttributeValueType {
                AVT_Numeric  = 1,    /* the original one */
                AVT_Complex  = 2
} DbvAttributeValueType;




          /* -------------   DbvIntUnit   -------------------- */

typedef enum DbvStringOrNumericType {
                 SNT_String  = 1,
                 SNT_Numeric = 2
} DbvStringOrNumericType;


typedef struct DbvStringOrNumeric {
DbvStringOrNumericType     eStringOrNumeric;
union {
    DbvInternationalString    *psString;
    Ulong                      ulNumeric;
  }                    uStringOrNumeric;
} DbvStringOrNumeric;


typedef struct DbvUnit {
DbvInternationalString *psUnitSystem;
DbvStringOrNumeric     *psUnitType;
DbvStringOrNumeric     *psUnit;
Ulong                  *pulScaleFactor;
} DbvUnit;
```

```
typedef struct DbvIntUnit {
Ulong          ulValue;
DbvUnit        sUnitUsed;
} DbvIntUnit;


typedef struct DbvComplexValue {
DbvStringOrNumeric    **ppsList;
Ulong                 **ppulSemanticAction;
} DbvComplexValue;


typedef struct DbvAttributeValue {
DbvAttributeValueType          eAttributeValueType;
union {
    Ulong                   ulNumeric;
    DbvComplexValue        *psComplexValue;
  }                            uAttributeValue;
} DbvAttributeValue;


typedef struct DbvAttributeElement {
DbvObjectIdentifier      pcAttributeSetId;
Ulong                    ulAttributeType;
DbvAttributeValue        sAttributeValue;
} DbvAttributeElement;


        /* -------------   DbvTerm   ------------------ */

typedef struct DbvTerm {
DbvTermType        eTermType;
union {
    DbvOctetString              *psGeneral;
    Ulong                        ulInteger;
    DbvInternationalString      *psSimpleCharacter;
    DbvObjectIdentifier          pcObjectIdentifier;
    DbvGeneralizedTime           pcDateTime;
    DbvASN1External             *psExternal;
    DbvIntUnit                  *psIntegerAndUnit;
  }               uTerm;
} DbvTerm;


typedef struct DbvAttributesPlusTerm {
DbvAttributeElement    **ppsAttributeElement;
DbvTerm                  sTerm;
} DbvAttributesPlusTerm;


typedef struct DbvResultSetPlusAttributes {
DbvInternationalString       sResultSetId;
DbvAttributeElement        **ppsAttributeElement;
} DbvResultSetPlusAttributes;


typedef struct DbvOperand {
```

```
DbvOperandType   eOperandType;
union {
    DbvAttributesPlusTerm      *psAttributesPlusTerm;
    DbvInternationalString     *psResultSetId;
    DbvResultSetPlusAttributes *psResultSetPlusAttributes;
  }            uOperand;
} DbvOperand;


typedef struct DbvRPNLopRopOperator {
struct DbvRPNStructure  *psLeftOperand;
struct DbvRPNStructure  *psRightOperand;
DbvOperator              sOperator;
} DbvRPNLopRopOperator;


typedef struct DbvRPNStructure {
DbvRPNStructureType   eRpnStructureType;
union {
    DbvRPNLopRopOperator       *psRpnLopRopOperator;
    DbvOperand                 *psRpnOperand;
  }                uRPNStructure;
} DbvRPNStructure;



typedef struct DbvRPNQuery {
DbvObjectIdentifier         pcAttributeSetId;
DbvRPNStructure             sRpnStructure;
} DbvRPNQuery;


typedef struct DbvQuery {
DbvQueryType        eQueryType;
union {
    DbvOctetString          *psPrivateQuery;
    DbvRPNQuery             *psRpnQuery;
    DbvOctetString          *psIso8777Query;
    DbvOctetString          *psZ3958Query;
    DbvRPNQuery             *psErpnQuery;
    DbvOctetString          *psNaturalLanguageQuery;
  }              uQuery;
} DbvQuery;




/* --------------  DbvRecords  ----------------------- */

typedef enum DbvRecordType {
              RT_Response                    = 1,
              RT_NonSurrDiagnostics          = 2,
              RT_MultipleNonSurrDiagnostics = 3
} DbvRecordType;


typedef enum DbvDbRecordType {
                DRT_Retrieval            = 1,
                DRT_SurrogateDiagnostics = 2,
```

```
                DRT_StartingFragment     = 3,
                DRT_IntermediateFragment = 4,
                DRT_FinalFragment        = 5
} DbvDbRecordType;


typedef enum DbvDiagnosticsType {
                   DT_DefaultFormat     = 1,
                   DT_ExternallyDefined = 2
} DbvDiagnosticsType;


typedef enum DbvFragmentSyntaxType {
                      FST_ExternallyTagged    = 1,
                      FST_NotExternallyTagged = 2
} DbvFragmentSyntaxType;


typedef enum DbvProtocolVersionInForce {
                      PVIF_Version2 = 1,
                      PVIF_Version3 = 2
} DbvProtocolVersionInForce;


typedef struct DbvAdditionalInfo {
DbvProtocolVersionInForce   eProtocolVersionInForce;
union {
    DbvVisibleString            pcV2;
    DbvInternationalString     *psV3;
  }                            uAdditionalInfo;
} DbvAdditionalInfo;

typedef struct DbvDefaultDiagFormat {
DbvObjectIdentifier        pcDiagnosticSetId;
Ulong                      ulCondition;
DbvAdditionalInfo          sAdditionalInfo;
} DbvDefaultDiagFormat;


typedef struct DbvDiagRec {
DbvDiagnosticsType       eDiagnosticsType;
union {
    DbvDefaultDiagFormat      *psDefault;
    DbvASN1External           *psExternallyDefined;
  }                  uDiagnostics;
} DbvDiagRec;


typedef struct DbvFragmentSyntax {
DbvFragmentSyntaxType      eFragmentSyntaxType;
union {
    DbvASN1External    *psExternallyTagged;
    DbvOctetString     *psNotExternallyTagged;
  }                     uFragmentSyntax;
} DbvFragmentSyntax;


typedef struct DbvNamePlusRecord {
DbvInternationalString  *psDatabaseName;
```

```
DbvDbRecordType          eDbRecordType;
union {
    DbvASN1External   *psRetrievalRecord;
    DbvDiagRec        *psSurrogateDiagnostic;
    DbvFragmentSyntax *psStartingFragment;
    DbvFragmentSyntax *psIntermediateFragment;
    DbvFragmentSyntax *psFinalFragment;
  }                   uDbRecord;
} DbvNamePlusRecord;



typedef struct DbvRecords {
DbvRecordType        eRecordType;
union {
    DbvNamePlusRecord    **ppsResponseRecord;
    DbvDefaultDiagFormat  *psNonSurrogateDiag;
    DbvDiagRec           **ppsMultipleNonSurrogateDiag;
  }               uRecord;
} DbvRecords;



/* --------------  DbvRange  ----------------------- */

typedef struct DbvRange {
Ulong        ulStartingPosition;
Ulong        ulNumberOfRecords;
} DbvRange;



/* --------------  RecordComposition  ----------------------- */


typedef enum DbvRecordCompositionType {
                         RCT_ElementSetNames = 1,
                         RCT_CompSpec        = 2
} DbvRecordCompositionType;


typedef enum DbvElementSpecType {
                   EST_Primitive = 1,
                   EST_External  = 2
} DbvElementSpecType;


typedef struct DbvSpecification {
DbvObjectIdentifier       pcSchema;
DbvElementSpecType        eElementSpecType;
union {
    DbvInternationalString  *psPrimitive;
    DbvASN1External         *psExternal;
  }                    uElementSpec;
} DbvSpecification;


typedef struct DbvDbSpecificComp {
DbvInternationalString    sDatabaseName;
```

```
DbvSpecification          sSpecification;
} DbvDbSpecificComp;


typedef struct DbvCompSpec {
DbvBool                   fSelectAlternativeSyntax;
DbvSpecification         *psGeneric;
DbvDbSpecificComp      **ppsDbSpecific;
DbvObjectIdentifier      *ppcRecordSyntax;
} DbvCompSpec;


typedef struct DbvRecordComposition {
DbvRecordCompositionType    eRecordCompositionType;
union {
    DbvElementSetNames  *psElementSetNames;
    DbvCompSpec         *psCompSpec;
  }                      uRecordComposition;
} DbvRecordComposition;



/* ----------  ListStatus  (--> DELETE-RESULT-SET response) --------- */

typedef enum DbvDeleteSetStatus {
                  DSS_Success                       = 0,
                  DSS_ResultSetDidNotExist          = 1,
                  DSS_PreviouslyDeletedByTarget     = 2,
                  DSS_SystemProblemAtTarget         = 3,
                  DSS_AccessNotAllowed              = 4,
                  DSS_ResourceControlAtOrigin       = 5,
                  DSS_ResourceControlAtTarget       = 6,
                  DSS_BulkDeleteNotSupported        = 7,
                  DSS_NotAllSetsDeletedOnBulkDlt    = 8,
                  DSS_NotAllRequestedSetsDeleted    = 9,
                  DSS_ResultSetInUse                = 10,
                  DSS_Invalid                       = -1
} DbvDeleteSetStatus;


typedef struct DbvListStatus {
DbvInternationalString       sResultSetId;
DbvDeleteSetStatus           eDeleteSetStatus;
} DbvListStatus;



/* --------------  DbvListEntries  (--> SCAN ) ---------------- */


typedef enum DbvEntryType {
             ET_TermInfo            = 1,
             ET_SurrogateDiagnostic = 2
} DbvEntryType;


typedef enum DbvOccurrenceType {
                  OT_Global     = 1,
                  OT_ByDatabase = 2
```

```
} DbvOccurrenceType;


typedef struct DbvOccurrenceByDb {
DbvInternationalString       sDatabaseName;
Ulong                       *pulNumber;
DbvOtherInformation        **ppsOtherInformation;
} DbvOccurrenceByDb;


typedef struct DbvOccurrence {
DbvOccurrenceType             eOccurrenceType;
union {
    Ulong                ulGlobal;
    DbvOccurrenceByDb  **ppsOccurrenceByDb;
  }                           uOccurence;
} DbvOccurrence;


typedef struct DbvOccurrenceByAttributes {
DbvAttributeElement      **ppsAttributeElement;
DbvOccurrence             *psOccurrence;
DbvOtherInformation      **ppsOtherInformation;
} DbvOccurrenceByAttributes;


typedef struct DbvTermInfo {
DbvTerm                    sTerm;
DbvInternationalString     *psDisplayTerm;
DbvAttributeElement       **ppsSuggestedAttribute;
DbvAttributesPlusTerm     **ppsAlternativeTerm;
Ulong                      *pulGlobalOccurrences;
DbvOccurrenceByAttributes **ppsByAttributes;
DbvOtherInformation       **ppsOtherInformation;
} DbvTermInfo;


typedef struct DbvEntry {
DbvEntryType         eEntryType;
union {
    DbvTermInfo    *psTermInfo;
    DbvDiagRec     *psSurrogateDiagnostic;
  }              uEntry;
} DbvEntry;


typedef struct DbvListEntries {
DbvEntry    **ppsEntry;
DbvDiagRec  **ppsNonSurrogateDiagnostics;
} DbvListEntries;



/* --------------  SortSequence  ----------------------- */

typedef enum DbvSortElementType {
                     SET_Generic    = 1,
                     SET_DbSpecific = 2
} DbvSortElementType;
```

```
typedef enum DbvSortRelationType {
                    SRT_Ascending     = 1,
                    SRT_Descending    = 2,
                    SRT_FrequencyAsc  = 3,
                    SRT_FrequencyDesc = 4,
                    SRT_Invalid       = -1
} DbvSortRelationType;


typedef enum DbvSortCaseType {
                SCT_CaseSensitive   = 1,
                SCT_CaseInsensitive = 2,
                SCT_Invalid         = -1
} DbvSortCaseType;


typedef enum DbvSortKeyType {
                    SKT_SortField      = 1,
                    SKT_ElementSpec    = 2,
                    SKT_SortAttributes = 3
} DbvSortKeyType;


typedef enum DbvSortMVActionType {
                    SMT_Abort    = 1,
                    SMT_Null     = 2,
                    SMT_Constant = 3
} DbvSortMVActionType;


typedef struct DbvSortAttributes {
DbvObjectIdentifier     pcAttributeSetId;
DbvAttributeElement   **ppsAttributeElement;
} DbvSortAttributes;


typedef struct DbvSortKey {
DbvSortKeyType              eSortKeyType;
union {
    DbvInternationalString     *psSortField;
    DbvSpecification           *psElementSpec;
    DbvSortAttributes          *psSortAttributes;
  }                     uSortElement;
} DbvSortKey;


typedef struct DbvSortMVAction {
DbvSortMVActionType    eSortMVActionType;
DbvOctetString        *psOctetString;  /* <--> eSortMVActionType == SMT_Constant
*/
} DbvSortMVAction;


typedef struct DbvDbSpecificSortKey {
DbvInternationalString      sDatabaseName;
DbvSortKey                  sDbSort;
} DbvDbSpecificSortKey;
```

```
typedef struct DbvSortElement {
DbvSortElementType        eSortElementType;
union {
    DbvSortKey               *psGeneric;
    DbvDbSpecificSortKey     **ppsDatabaseSpecific;
  }                       uSortElement;
} DbvSortElement;


typedef struct DbvSortKeySpec {
DbvSortElement         sSortElement;
DbvSortRelationType    eSortRelationType;
DbvSortCaseType        eSortCaseType;
DbvSortMVAction       *psMVActionType;
} DbvSortKeySpec;


/* ---  OriginSuppliedCommonParameters  (--> EXTENDED-SERVICES) ----------- */

typedef enum DbvPermAllowableFunctions {
                      PAF_Delete            = 1,
                      PAF_ModifyContents    = 2,
                      PAF_ModifyPermissions = 3,
                      PAF_Present           = 4,
                      PAF_Invoke            = 5,
                      PAF_Invalid           = -1
} DbvPermAllowableFunctions;


typedef struct DbvPermissions {
DbvInternationalString          sUserId;
DbvPermAllowableFunctions     **ppeAllowableFunctions;
} DbvPermissions;
```

B.4 Object Identifier, Record Syntaxes, Resource Report, Update Extended Service

```
/* ------------------------------------------------------------------*/
/* Syntax of Object Identifier definitions:

   ObjectIdentifier ::= Header Trailer
   Header ::= "1.0.10163." | "1.2.840.10003."
   Trailer ::= Number"."Number"}"
   Number  ::= positive integer value



 ------------------------------------------------------------------*/
```

```
/* -------------- Object Identifier (Record Syntaxes) -----------*/


#define ISO_UNIMARC             "1.0.10163.5.1"
#define ANSI_UNIMARC            "1.2.840.10003.5.1"
#define ISO_INTERMARC           "1.0.10163.5.2"
#define ANSI_INTERMARC          "1.2.840.10003.5.2"
#define ISO_CCF                 "1.0.10163.5.3"
#define ANSI_CCF                "1.2.840.10003.5.3"
#define ISO_USMARC              "1.0.10163.5.10"
#define ANSI_USMARC             "1.2.840.10003.5.10"
#define ISO_UKMARC              "1.0.10163.5.11"
#define ANSI_UKMARC             "1.2.840.10003.5.11"
#define ISO_NORMARC             "1.0.10163.5.12"
#define ANSI_NORMARC            "1.2.840.10003.5.12"
#define ISO_LIBRISMARC          "1.0.10163.5.13"
#define ANSI_LIBRISMARC         "1.2.840.10003.5.13"
#define ISO_DANMARC             "1.0.10163.5.14"
#define ANSI_DANMARC            "1.2.840.10003.5.14"
#define ISO_FINMARC             "1.0.10163.5.15"
#define ANSI_FINMARC            "1.2.840.10003.5.15"
#define ISO_MAB                 "1.0.10163.5.16"
#define ANSI_MAB                "1.2.840.10003.5.16"
#define ISO_CANMARC             "1.0.10163.5.17"
#define ANSI_CANMARC            "1.2.840.10003.5.17"
#define ISO_SBN                 "1.0.10163.5.18"
#define ANSI_SBN                "1.2.840.10003.5.18"
#define ISO_PICAMARC            "1.0.10163.5.19"
#define ANSI_PICAMARC           "1.2.840.10003.5.19"
#define ISO_AUSMARC             "1.0.10163.5.20"
#define ANSI_AUSMARC            "1.2.840.10003.5.20"

#define ISO_UNSTRUCTURED_TEXT_RECORD   "1.0.10163.5.101"
#define ANSI_UNSTRUCTURED_TEXT_RECORD  "1.2.840.10003.5.101"
#define ISO_OPAC_RECORD                "1.0.10163.5.102"
#define ANSI_OPAC_RECORD               "1.2.840.10003.5.102"
#define ISO_SUMMARY_RECORD             "1.0.10163.5.103"
#define ANSI_SUMMARY_RECORD            "1.2.840.10003.5.103"
#define ISO_ES_TASK_PACKAGE_RECORD     "1.0.10163.5.106"
#define ANSI_ES_TASK_PACKAGE_RECORD    "1.2.840.10003.5.106"



/* -------------- Object Identifier (Attribute-Set) -----------*/

#define ISO_ATTRIBUTE_SET_BIB_1      "1.0.10163.3.1"
#define ANSI_ATTRIBUTE_SET_BIB_1     "1.2.840.10003.3.1"




/* -------------- Object Identifier (Diagnostic-Set) -----------*/

#define ISO_DIAGNOSTIC_SET_BIB_1      "1.0.10163.4.1"
#define ANSI_DIAGNOSTIC_SET_BIB_1     "1.2.840.10003.4.1"
```

```
/* -------------- Object Identifier (Resource-Report) -----------*/

#define ISO_RESOURCE_REPORT_1        "1.0.10163.7.1"
#define ANSI_RESOURCE_REPORT_1       "1.2.840.10003.7.1"
#define ISO_RESOURCE_REPORT_2        "1.0.10163.7.2"
#define ANSI_RESOURCE_REPORT_2       "1.2.840.10003.7.2"



/* -------------- Object Identifier (UPDATE Extended Service) -----------*/

#define ISO_UPDATE_EXTENDED_SERVICE        "1.0.10163.9.5"
#define ANSI_UPDATE_EXTENDED_SERVICE       "1.2.840.10003.9.5"




/* -------------- Record Syntax "UnstructuredText" ----------------*/

typedef DbvInternationalString   DbvUnstructuredText;



/* -------------- Record Syntax "OpacRecord" ----------------*/

typedef struct DbvCircRecord {
DbvBool                      fAvailableNow;
DbvInternationalString     *psAvailabilityDate;
DbvInternationalString     *psAvailableThru;
DbvInternationalString     *psRestrictions;
DbvInternationalString     *psItemId;
DbvBool                      fRenewable;
DbvBool                      fOnHold;
DbvInternationalString     *psEnumAndChron;
DbvInternationalString     *psMidspine;
DbvInternationalString     *psTemporaryLocation;
} DbvCircRecord;


typedef struct DbvVolume {
DbvInternationalString        *psEnumeration;
DbvInternationalString        *psChronology;
DbvInternationalString        *psEnumAndChron;
} DbvVolume;


typedef struct DbvHoldingsAndCircData {
DbvInternationalString        *psTypeOfRecord;
DbvInternationalString        *psEncodingLevel;
DbvInternationalString        *psFormat;
DbvInternationalString        *psReceiptAcqStatus;
DbvInternationalString        *psGeneralRetention;
DbvInternationalString        *psCompleteness;
```

```
DbvInternationalString        *psDateOfReport;
DbvInternationalString        *psNucCode;
DbvInternationalString        *psLocalLocation;
DbvInternationalString        *psShelvingLocation;
DbvInternationalString        *psCallNumber;
DbvInternationalString        *psShelvingData;
DbvInternationalString        *psCopyNumber;
DbvInternationalString        *psPublicNote;
DbvInternationalString        *psReproductionNote;
DbvInternationalString        *psTermsUseRepro;
DbvInternationalString        *psEnumAndChron;
DbvVolume                    **ppsVolume;
DbvCircRecord                **ppsCircRecord;
} DbvHoldingsAndCircData;



typedef enum DbvHoldingsRecType {
      HRT_MarcHoldingsRecord  = 1,
      HRT_HoldingsAndCirc     = 2
} DbvHoldingsRecType;


typedef struct DbvHoldingsRecord {
DbvHoldingsRecType      eHoldingsRecType;
union {
    DbvASN1External         sMarcHoldingsRecord;
    DbvHoldingsAndCircData  sHoldingsAndCirc;
  }                    uMarcOrHoldAndCirc;
} DbvHoldingsRecord;


typedef struct DbvOpacRecord {
DbvASN1External         *psBibliographicRecord;
DbvHoldingsRecord      **ppsHoldingsData;
} DbvOpacRecord;



/* -------------- Record Syntax "SummaryRecord" -----------------*/

typedef struct DbvFormatSpec {
DbvInternationalString          sType;
Ulong                          *pulSize;
Ulong                          *pulBestPosn;
} DbvFormatSpec;



typedef struct DbvSummaryRecord {
DbvInternationalString           sTitle;
DbvInternationalString          *psAuthor;
DbvInternationalString          *psCallNumber;
DbvInternationalString          *psRecordType;
DbvInternationalString          *psBibliographicLevel;
DbvFormatSpec                   **ppsFormat;
DbvInternationalString          *psPublicationPlace;
DbvInternationalString          *psPublicationDate;
DbvInternationalString          *psTargetSystemKey;
```

```
DbvInternationalString          *psSatisfyingElement;
Ulong                           *pulRank;
DbvInternationalString          *psDocumentId;
DbvInternationalString          *psAbstract;
DbvOtherInformation            **ppsOtherInformation;
} DbvSummaryRecord;



/* -------------- Record Syntax "Resource Report" ----------------*/

typedef enum DbvEstimateType {
                ET_CurrentSearchRecords    = 1,
                ET_FinalSearchRecords      = 2,
                ET_CurrentPresentRecords   = 3,
                ET_FinalPresentRecords     = 4,
                ET_CurrentOpTimeProcessing = 5,
                ET_FinalOpTimeProcessing   = 6,
                ET_CurrentAssocTime        = 7,
                ET_CurrentOperationCost    = 8,
                ET_FinalOperationCost      = 9,
                ET_CurrentAssocCost        = 10,
                ET_FinalOpTimeElapsed      = 11,
                ET_PercentComplete         = 12,
                ET_CurrentSearchAssocCost  = 13,
                ET_CurrentPresentAssocCost = 14,
                ET_CurrentConnectAssocCost = 15,
                ET_CurrentOtherAssocCost   = 16,
                ET_Invalid                 = -1
} DbvEstimateType;


typedef struct DbvEstimate1 {
DbvEstimateType          eEstimateType;
Ulong                    ulValue;
Ulong                   *pulCurrencyCode;
} DbvEstimate1;


typedef struct DbvResourceReport1 {
DbvEstimate1                **ppsEstimate;
DbvInternationalString       sMessage;
} DbvResourceReport1;


typedef struct DbvEstimate2 {
DbvStringOrNumeric       sType;
DbvIntUnit               sValue;
} DbvEstimate2;

typedef struct DbvResourceReport2 {
DbvEstimate2               **ppsEstimate;
DbvInternationalString       sMessage;
} DbvResourceReport2;
```

```
/* -------------- Record Syntax "Extended Services Task Package" ------------*/


typedef enum DbvExtServicesTaskStatus {
                ESTS_Pending  = 1,
                ESTS_Active   = 2,
                ESTS_Complete = 3,
                ESTS_Aborted  = 4,
                ESTS_Invalid  = -1
} DbvExtServicesTaskStatus;



typedef struct DbvEsTaskPackage {
DbvObjectIdentifier                  pcPackageType;
DbvInternationalString              *psPackageName;
DbvInternationalString              *psUserId;
DbvIntUnit                          *psRetentionTime;
DbvPermissions                     **ppsPermissions;
DbvInternationalString              *psDescription;
DbvOctetString                      *psTargetReference;
DbvGeneralizedTime                   pcCreationDateTime;
DbvExtServicesTaskStatus             eExtServicesTaskStatus;
DbvDiagRec                         **ppsPackageDiagnostics;
DbvASN1External                      sTaskSpecificParameters;
} DbvEsTaskPackage;




/* -------------  UPDATE  EXTENDED SERVICE ---------------- */

typedef enum DbvUpdtAction {
                        UA_RecordInsert  = 1,
                        UA_RecordReplace = 2,
                        UA_RecordDelete  = 3,
                        UA_ElementUpdate = 4,
                        UA_Invalid       = -1
} DbvUpdtAction;



typedef enum DbvUpdtTaskStatus {
                UTS_Success  = 1,
                UTS_Partial  = 2,
                UTS_Failure  = 3,
                UTS_Invalid  = -1
} DbvUpdtTaskStatus;


typedef enum DbvUpdtRecordStatus {
                URST_Success   = 1,
                URST_Queued    = 2,
                URST_InProcess = 3,
                URST_Failure   = 4,
                URST_Invalid   = -1
} DbvUpdtRecordStatus;
```

```
typedef enum DbvUpdtRecordOrDiagType {
                 URDT_Record     = 1,
                 URDT_Diagnostic = 2
} DbvUpdtRecordOrDiagType;


typedef enum DbvUpdtSupplementalIdType {
                USIT_Time            = 1,
                USIT_Version         = 2,
                USIT_PreviousVersion = 3
} DbvUpdtSupplementalIdType;


typedef enum DbvUpdtRecordIdType {
                URIT_Number = 1,
                URIT_String = 2,
                URIT_Opaque = 3
} DbvUpdtRecordIdType;


typedef struct DbvUpdtRecordOrDiag {
DbvUpdtRecordOrDiagType      eUpdtRecordOrDiag;
union {
    DbvASN1External      *psRecord;
    DbvDiagRec           *psDiagnostic;
  }                          uUpdtRecordOrDiag;
} DbvUpdtRecordOrDiag;


typedef struct DbvCorrelationInfo {
DbvInternationalString  *psNote;
Ulong                   *pulId;
} DbvCorrelationInfo;


typedef struct DbvTaskPackageRecord {
DbvUpdtRecordOrDiag       *psUpdtRecordOrDiag;
DbvCorrelationInfo        *psCorrelationInfo;
DbvUpdtRecordStatus        eUpdtRecordStatus;
} DbvTaskPackageRecord;


typedef struct DbvUpdateTargetPart {
DbvUpdtTaskStatus           eTaskStatus;
DbvDiagRec                **ppsGlobalDiagnostics;
DbvTaskPackageRecord      **ppsTaskPackageRecord;
} DbvUpdateTargetPart;


typedef struct DbvUpdtSupplementalId {
DbvUpdtSupplementalIdType     eSupplementalIdType;
union {
    DbvGeneralizedTime      pcTime;
    DbvInternationalString *psVersion;
    DbvASN1External        *psPreviousVersion;
  }                           uUpdtSupplementalId;
} DbvUpdtSupplementalId;
```

```
typedef struct DbvUpdtRecordId {
DbvUpdtRecordIdType   eRecordIdType;
union {
    Ulong                    ulNumber;
    DbvInternationalString *psString;
    DbvOctetString         *psOpaque;
  }                   uUpdtRecordId;
} DbvUpdtRecordId;


typedef struct DbvSuppliedRecord {
DbvUpdtRecordId            *psUpdtRecordId;
DbvUpdtSupplementalId      *psUpdtSupplementalId;
DbvCorrelationInfo         *psCorrelationInfo;
DbvASN1External             sRecord;
} DbvSuppliedRecord;


typedef struct DbvOriginPartToKeep {
DbvUpdtAction               eAction;
DbvInternationalString      sDatabaseName;
DbvObjectIdentifier         pcSchema;
DbvInternationalString     *psElementSetName;
} DbvOriginPartToKeep;


typedef struct DbvUpdateTaskPackage {
DbvOriginPartToKeep      sOriginPartToKeep;
DbvUpdateTargetPart      sTargetPart;
} DbvUpdateTaskPackage;


typedef struct DbvUpdateRequestPackage {
DbvOriginPartToKeep    sOriginPartToKeep;
DbvSuppliedRecord    **ppsSuppliedRecord;
} DbvUpdateRequestPackage;


typedef enum DbvUpdateReqOrRsp {
                    UROR_Request  = 1,
                    UROR_Response = 2
} DbvUpdateReqOrRsp;


typedef struct DbvUpdateExtendedService {
DbvUpdateReqOrRsp     eUpdateReqRsp;
union {
    DbvUpdateRequestPackage  *psUpdateRequest;
    DbvUpdateTaskPackage     *psUpdateResponse;
  }                uUpdateReqOrRsp;
} DbvUpdateExtendedService;
```