Unit 3, The Paddock
Hambridge Road
Newbury
Berkshire
United Kingdom

Project: DBV-OSI II

Project: ONE – OPAC Network in Europe

-Z39.50 API-Definition Manual

Document No.: D-008
Part A – Z39.50 API Descriptions

Document Version: 7.0

Z39.50 API Software Version 2.1

Prepared by:	R. A. Bull	Crossnet
Based on Original by:	B. Hergeth	Danet GmbH

This document is based upon the DBV OSI II API definition Manual d-008 Version 5, prepared by B. Hergeth, DANET GmbH for the DBV OSI II Project.

Extensions to the DBV OSI II API have been added under the auspices of the ONE project. These extensions are included in this document and follow the style of the original design.

Table of Contents

1. Introduction....6
1.1 Scope....6
1.2 Communication Stacks....6

2. General description of the API....8
2.1 General Description of the API Functions....8
2.2 Naming Conventions....9
2.3 Organisation of API Functions....10

3. API-Functions....11
3.1 Functions Used by the Origin System....11
3.1.1 Association Handling Functions Using the ACSE-Services....11
3.1.1.1 DbvAssociateRequest()....12
3.1.1.2 DbvReleaseRequest()....18
3.1.1.3 DbvAbortRequest()....21
3.1.1.4 DbvConfirmAssociateRequest()....23
3.1.1.5 DbvConfirmReleaseRequest()....27
3.1.2 Association Handling Functions Using TCP-Services....29
3.1.2.1 DbvAssociateRequest()....29
3.1.2.2 DbvReleaseRequest()....33
3.1.2.3 DbvAbortRequest()....35
3.1.2.4 DbvConfirmAssociateRequest()....37
3.1.2.5 DbvConfirmReleaseRequest()....37
3.1.3 SR-/Z39.50 Functions....38
3.1.3.1 DbvInitializeRequest()....38
3.1.3.2 DbvSearchRequest()....40
3.1.3.3 DbvPresentRequest()....41
3.1.3.4 DbvDeleteResultSetRequest()....42
3.1.3.5 DbvScanRequest()....43
3.1.3.6 DbvResourceReportRequest()....44
3.1.3.7 DbvSortRequest()....45
3.1.3.8 DbvExtendedServicesRequest()....46
3.1.3.9 DbvCloseRequest()....47
3.1.3.10 DbvCloseResponse()....48
3.1.3.11 DbvAccessControlResponse()....49
3.1.3.12 DbvResourceControlResponse()....50
3.1.3.13 DbvTriggerResourceControlRequest()....51
3.1.3.14 DbvReceiveDataOrigin()....52
3.2 Functions Used by the Target System....54
3.2.1 DbvTargetInitialize()....54
3.2.2 Association Handling Functions Using the ACSE-Services....55
3.2.2.1 DbvReceiveAssociateRequest()....56
3.2.2.2 DbvAssociateResponse()....60
3.2.2.3 DbvReleaseResponse()....63
3.2.2.4 DbvAbortRequest()....65
3.2.3 Association Handling Functions Using TCP-Services....66

3.2.3.1 DbvReceiveAssociateRequest() ... 66
3.2.3.2 DbvAssociateResponse() ... 69
3.2.3.3 DbvReleaseResponse() ... 71
3.2.3.4 DbvAbortRequest() ... 73
3.2.4 SR-/Z39.50 Functions ... 74
3.2.4.1 DbvInitializeResponse() ... 74
3.2.4.2 DbvSearchResponse() ... 76
3.2.4.3 DbvPresentResponse() ... 78
3.2.4.4 DbvDeleteResultSetResponse() ... 79
3.2.4.5 DbvScanResponse() ... 80
3.2.4.6 DbvResourceReportResponse() ... 81
3.2.4.7 DbvSortResponse() ... 82
3.2.4.8 DbvSegmentationRequest() ... 83
3.2.4.9 DbvExtendedServicesResponse() ... 84
3.2.4.10 DbvCloseRequest() ... 85
3.2.4.11 DbvCloseResponse() ... 86
3.2.4.12 DbvAccessControlRequest() ... 87
3.2.4.13 DbvResourceControlRequest() ... 88
3.2.4.14 DbvReceiveDataTarget() ... 89
3.3 Utility Functions ... 91
3.3.1 SIGetEventLocation() ... 91
3.3.2 SISetACSEFile() ... 94
3.3.3 SIWhichStack() ... 96
3.3.4 SISetTraceParameter() ... 99
3.4 Utility Functions to Release DBV Datastructure Memory ... 101
3.4.1 Releasing Memory in a PDU Received at the Origin ... 101
3.4.1.1 SIFreeDbvOriginData() ... 101
3.4.2 Releasing Memory for the Origin SR/Z39.50 Functions ... 103
3.4.2.1 SIFreeDbvInitializeReq() ... 103
3.4.2.2 SIFreeDbvSearchReq() ... 105
3.4.2.3 SIFreeDbvPresentReq() ... 107
3.4.2.4 SIFreeDbvDeleteResultSetReq() ... 109
3.4.2.5 SIFreeDbvScanReq() ... 111
3.4.2.6 SIFreeDbvResourceReportReq() ... 113
3.4.2.7 SIFreeDbvSortReq() ... 115
3.4.2.8 SIFreeDbvExtendedServicesReq() ... 117
3.4.2.9 SIFreeDbvCloseReq() ... 119
3.4.2.10 SIFreeDbvAccessControlRsp() ... 121
3.4.2.11 SIFreeDbvResourceControlRsp() ... 123
3.4.2.12 SIFreeDbvTriggerResourceControlReq() ... 125
3.4.3 Releasing Memory in a PDU Received at the Target ... 127
3.4.3.1 SIFreeDbvTargetData() ... 127
3.4.4 Releasing Memory for the Target SR/Z39.50 Functions ... 129
3.4.4.1 SIFreeDbvInitializeRsp() ... 129
3.4.4.2 SIFreeDbvSearchRsp() ... 131
3.4.4.3 SIFreeDbvPresentRsp() ... 133
3.4.4.4 SIFreeDbvDeleteResultSetRsp() ... 135
3.4.4.5 SIFreeDbvScanRsp() ... 137
3.4.4.6 SIFreeDbvResourceReportRsp() ... 139
3.4.4.7 SIFreeDbvSortRsp() ... 141
3.4.4.8 SIFreeDbvExtendedServicesRsp() ... 143
3.4.4.9 SIFreeDbvAccessControlReq() ... 145

3.4.4.10 SIFreeDbvResourceControlReq() 147

Annex A: Reference List **149**

Annex B: The Default "ACSE-configuration-file" 150

Table of Figures

1 TCP/IP and OSI Stacks 7

2 Structure of API Libraries 10

1. Introduction

1.1 Scope

This document is the API (Application Programmers Interface) manual for the DBV OSI II API software. The software is a Freeware package for the UNIX operating system that allows developers to write Z39.50 client and server applications.

The API software was been originally developed for the German Library Project, DBV OSI II, and has been subsequently extended in the scope of the European Project ONE – OPAC Network in Europe. The API software has been developed by Crossnet Systems Limited.

This document is provided in two parts:

Part A provides a general description of the API and a description of the API functions.

Part B provides information on the API data structures.

This document supports release 2.1 of the API software. The main additions to the API for this version are the support for the Character Set and Language Negotiation record, and the inclusion of memory free functions to free memory allocated for the API data structures.

1.2 Communication Stacks

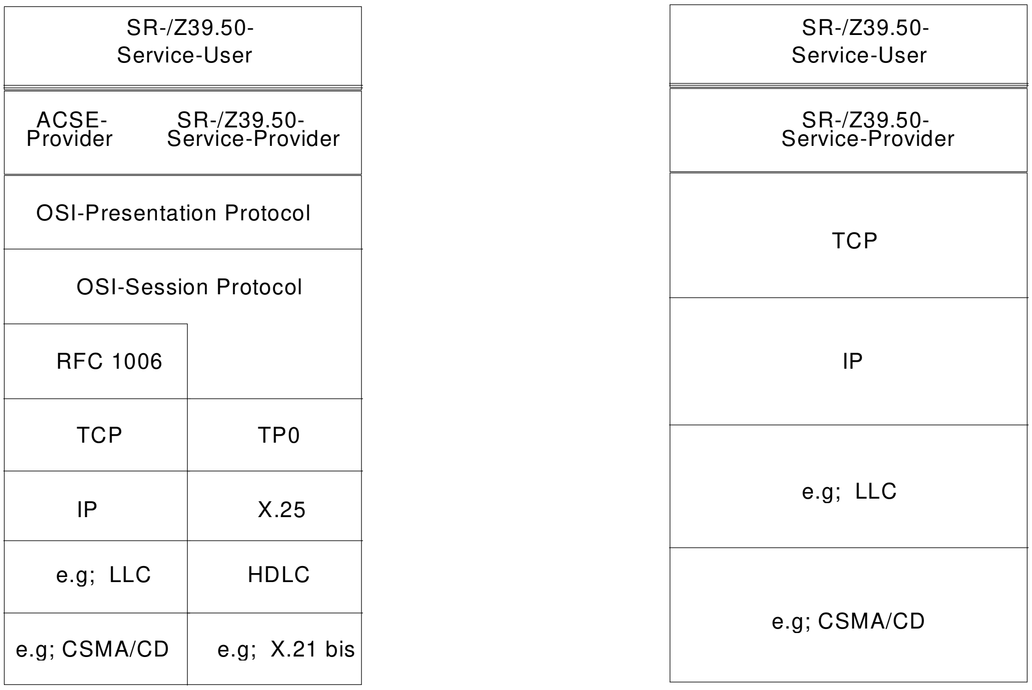
The Z39.50 API realises the interface between the "SR-/Z39.50 service-user" and the "SR-/Z39.50 service-provider"

The Z39.50 API software supports two protocol stacks:

- the OSI stack as shown in part 1 of the diagram below;
- the "Transmission Control Protocol" (TCP) stack as shown in part 2 of the diagram below.

In most computer systems, the UNIX operating system contains support for the TCP/IP stack.

Support for the OSI stack usually necessitates a vendor specific layered product or a third party product. The API has been successfully tested with the ISODE OSI stack product.



Part 1
OSI Stack Model

Part 2
TCP Stack Model

Figure 1 - TCP/IP and OSI Stacks

The API Functions are divided into:

- Functions to establish, release and abort connections (associations);
- Functions for the realisation of the main SR-/Z39.50 data transfer; and,
- Functions to release memory allocated for the API data structures.

In the view of the API user, the behaviour of the functions for the realisation of the main data transfers is independent of the communications environment.

For the Functions that realise the association handling, specific functions exist for the particular stack used. In Chapter 3, the functions are categorised by association, data transfer and utility functions. For Target applications that can be invoked by more than one stack type, section 3.3.3, the SIWhichStack()) function, can be used to identify which stack is in use.

The general description of the API is given in Chapter 2.

In Chapter 3 the API functions are described in detail. The data structures that are referred to in the API functions are described in Part 2 of this document.

2. General description of the API

This Chapter gives a general description of the API.

Sections 2.1 and 2.2 describe the general characteristics of the functions defined in Chapter 3 and explain the naming conventions used within the definition of the function parameters. Section 2.3 describes the form of the API functions required by the user.

2.1 General Description of the API Functions

The API provides:

- Functions for the establishment, release or abort of connections (associations);
- Functions for the realisation of SR/Z39.50 data transfer; and
- Some utility functions which are not communication orientated (utility functions, memory freeing functions)

Every API function replies a "Return Value" being either **SI_OK** or **SI_NOTOK**.

The value SI_OK indicates that the function was carried out without faults.

The value SI_NOTOK indicates that a fault occured during the excecution of the function. The fault that occurred will be shown in a data structure which is defined as follows:

```
typedef struct DbvError {
  DbvErrorClass         eSIErrorClass;
  union {
    DbvErrorCode        lIfOsCoError;
    DbvAbort            *psSIAbort;
  }         uErrorCode
  DbvAssociationState   eAssociationState
} SIError;
```

The Parameter *eSIErrorClass* indicates the fault class:

SI_Communication_Error	e.g, faulty PDU, disconnection by break-off etc.
SI_Interface_Error	e.g, function was called with invalid parameters etc.,
SI_OperatingSystem_Error	e.g, a data file could not be opened, insufficient memory, etc.,

SI_Abort	the association to the remote system no longer exists.

In the case of SI_Interface_Error, SI_Communication_Error and SI_OperatingSystem_Error, the "union" component *lIfOsCoError* will show the exact error.

In case of SI_Abort, the error will be indicated in the SIAbort data structure.

The parameter *eAssociationState* indicates if a connection to the partner system still exists.

The function parameter are labelled in the function description with "IN" or "OUT_C" or "OUT_F", meaning:

IN: A value of this parameter has to be transferred to the called function.

OUT_C: The function gives a value to this parameter. The memory for the value has to be provided by the function caller.

OUT_F: The function gives a value to the parameter. The memory for this value is provided by the function (system call malloc). The caller of the function is reponsible for the freeing of the memory (system call free).

2.2 Naming Conventions

The following conventions are used for the labelling of constants, data structures, elements of enumeration types and variables during definition:

Constants:	Every character is taken from {A...Z,_} (e.g. SI_NOTOK)
Names of structured data types:	Every character is taken from {A...Z,a...z}, the first character is always capital (e.g. SearchReq);
Elements of "enumeration types":	Every character is taken from {A...Z,a..z,_,1..9}, the first character is a capital (e.g. Association_Accepted).
Variables:	Every character is taken from {A...Z,a..z}, the first character is a lower case letter (e.g. psSearchReq).

Another rule is applied for the labelling of variables:

```
long     lAssocId;     /* l: long Variable */
long     *plAssocId;   /* pl: Pointer to a long Variable */
```

```
DbvErrorClass    eErrorClass;     /* e: "enumeration type" */

SearchReq        *psSearchReq;    /* ps: Pointer to a DataStructure */

DbvError         **ppsSIError;    /* pps: Pointer to a Pointer to a DataStructure */
```

2.3 Organisation of API Functions

The API functions are supplied in the form of function libraries. The linking to these libraries for the Origin and Target applications has to be done by the application developer. This process can be assisted by using Makefiles.

A stack independant API function is provided for every SR/Z39.50 service supported. This function in turn utilises the internal calling process to the APDU encoding/decoding operations and then calls the TCP or OSI specific functions.

The structure of the function libraries appears the same to the application developer for either stack. The OSI and TCP specific functions are called by the stack independent functions.

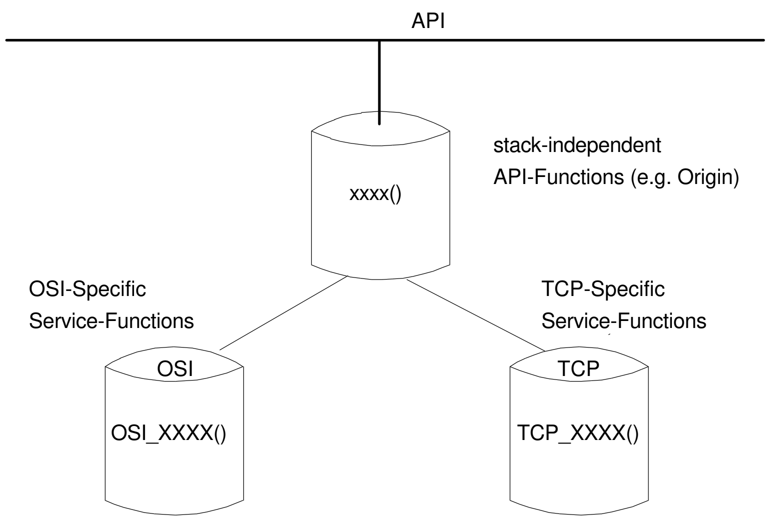


Figure 2 - Structure of API Libraries

3. API-Functions

This chapter describes the DBV-OSI II API Functions.

There is one set of API-functions defined for both implementations of the SR-/Z39.50 service provider

- the protocol conformant implementation, using the ACSE- and Presentation services
- the non-conformant implementation, i.e. the SR-/Z39.50 service provider is layered directly on top of TCP/IP.

From the API users point of view, the behaviour of the functions provided for the transfer of SR-/Z39.50 data is exactly the same in both environments.

The behaviour of the functions provided for association/connection handling is slightly different in the different environments. Some of the result values provided by these functions are permitted only in one of both environments. This requires an environment dependent handling of the result values.

For this reason, the association handling functions are described for both, when using the ACSE services and when using the TCP services.

Chapter 3.1 describes the API functions available to the "Origin System".

Chapter 3.2 describes the API functions available to the "Target System".

Chapter 3.3 describes some additional utility functions.

Chapter 3.4 describes the memory freeing utility functions.

3.1 Functions Used by the Origin System

Chapters 3.1.1 and 3.1.2 describe the association handling functions available to the API-user in the OSI-, and TCP-environment respectively.

Chapter 3.1.3 describes the functions available for sending and receiving SR-/Z39.50 data in both environments.

3.1.1 Association Handling Functions Using the ACSE-Services

The following functions are available to the API-user:

- DbvAssociateRequest()
- DbvReleaseRequest()
- DbvAbortRequest()
- DbvConfirmAssociateRequest()

- DbvConfirmReleaseRequest()

Chapters 3.1.1.1 - 3.1.1.5 describe these functions.

3.1.1.1 DbvAssociateRequest()

Purpose:

Establish association using the ACSE-service AAssociate.Request.

Function Name and Parameters:

```
DbvAssociateRequest (
DbvAssocId            *plAssocId,              /* OUT_C */
DbvSecs                lSeconds,               /* IN */
DbvTarget             *psTargetIdentification, /* IN */
DbvAssociateState     *peAssociateState,       /* OUT_C */
DbvAssociateResult   **ppuAssociateResult,     /* OUT_F */
DbvError             **ppsSIError              /* OUT_F */
}
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

plAssocId — A pointer to a long value.
In case the function returns SI_OK and the association was either accepted or the associate request is still pending, the value is updated and provides the unique identifier to be used to subsequently refer to the association.
In case the function returns SI_NOTOK or the association was rejected, the value is unchanged.

lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately. If the function returns SI_OK, then the value parameter *plAssocId* refers to, is the unique identifier which is to be used in subsequent calls to function DbvConfirmAssociateRequest() (refer to 3.1.1.4) or any other function provided to the API-user. The value, which parameter *peAssociateState* refers to, most probably will indicate ***Associate_Pending***. Note however, that the value may also indicate *Associate_Accepted* or *Associate_Rejected*, in which case the association establishment already terminated. A value ***CALL_BLOCKING*** causes the function to wait for either the expected response or an event indicating that the association could not be established. If the function returns SI_OK, the association was either established, or rejected by the remote user or the remote ACSE-provider. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of the expected response (or any other event indicating that the association could not be established). If the function returns SI_OK, the value parameter *peAssociateState* refers to indicates the state of the association. If the state is ***Associate_Pending***, i.e. the response had not yet been received, use either function DbvAbortRequest() (refer to 3.1.1.3) to abort the association establishment or use function DbvConfirmAssociateRequest() (refer to 3.1.1.4).
psTargetIdentification	A pointer to a value of type *DbvTarget*. The value provides either the name or the address of the target system, the association has to be established to.
peAssociateState	A pointer to a value of type *DbvAssociateState*. The value should be ***Associate_Null*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value is either ***Associate_Accepted*** or ***Associate_Rejected*** or ***Associate_Pending***. ***Associate_Accepted*** indicates, that the association was established. ***Associate_Rejected*** indicates, that the associate request was rejected by either the remote user or the remote ACSE-provider. ***Associate_Pending*** indicates, that no response arrived and no other event occured until now. This value may occur only if the actual value of lSeconds is ***CALL_ASYNCHRONOUS*** or a value greater than zero.

ppuAssociateResult	The address of a pointer of type **DbvAssociateResult*. When calling the function, the pointer should refer to (DbvAssociateResult*)NULL. The pointer is updated only if the function returns SI_OK and the value which parameter *peAssociateState* refers to either indicates ***Associate_Accepted*** or ***Associate_Rejected***. If the value which parameter *peAssociateState* refers to indicates ***Associate_Accepted*** the pointer refers to a *DbvProtocolAndSyntaxes* structure indicating the application protocol to be used over the association and the abstract syntaxes (e.g. UNIMARC, MAB, ResourceReportFormat Resource-1) commonly supported by the origin and the target system. If the value which parameter *peAssociateState* refers to indicates ***Associate_Rejected***, the pointer refers to a *DbvAssociateReject* structure indicating the reason for the rejection.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psTargetIdentification* the name or the address of the target system the association has to be established to.

In case the target name is provided, the function extracts the corresponding target address from a specific address file.

The function reads the data required to form a valid AAssociate.Request service primitive from the "ACSE-configuration-file" (refer to Annex B and see the notes below).

The function checks the data for completeness and presents the service primitive to the ACSE service provider. The ACSE service provider submits an AARQ-APDU.

If the function returns SI_NOTOK, an error occured and the association was most probably not established. Component *eAssociationState* of the error structure gives the exact answer.

The meaning of a return value SI_OK depends on the actual value of lSeconds (see description above).

If the association was established or the associate request is still pending, the value which parameter *plAssocId* refers to is the unique identifier of the association. This identifier is to be used in all subsequent function calls to any association handling and/or SR-/Z39.50 function.

If the association was established, but the application on top of the API does not support the protocol to be used then the association should be released immediately by calling function DbvReleaseRequest().

Notes:

The "ACSE-configuration-file" (refer to Annex B) provides values for the parameters "Application Context Name" and "Presentation Context Definition List" of the AAssociate.Request service primitive.

Note on "Application Context Name": The application on top of the API may support both protocols, SR and Z39.50. In this case there will be two values provided for "Application Context Name": "Basic_SR_Application_Context" and "Basic_Z3950_Application_Context". To fill-in the component of the AAssociate.Request service primitive, the function selects the first value appearing in the sequence.

Upon receiving the AAssociate.Confirmation service primitive, the function inspects the value of "Application Context Name" of the service primitive. If the value is one of "Basic_SR_Application_Context" or "Basic_Z3950_Application_Context", the function assignes the value "SR_Protocol", "Z3950_Protocol" respectively, to the component *peApplicationProtocol* of the *DbvProtocolAndSyntaxes* structure. If the value provided in "Application Context Name" indicates another value, the function assignes the value "Unidentified_Protocol" to that component. In this case, using function DbvReleaseRequest(), the API-user should immediately release the application association.

Notes on "Presentation Context Definition List": This is an optional parameter. If present, the paramter indicates the abstract syntaxes the originator of the association intends to use over the association. The user at the target system (and the Presentation provider) accepts or rejects each abstract syntax proposed by the originator of the association. The result is provided to the originator in parameter "Presentation Context Definition Result List" of the AAssociate.Confirmation service primitive.

Since the record syntaxes (UNIMARC, MAB etc.) and the format syntaxes (e.g. ResourceReportFormat Resource-1 etc.) are abstract syntaxes, the above parameters of the service primitives allow to determine whether there exist syntaxes which are commonly supported by both systems.

The value to be provided in the ACSE-configuration-file is a list of Object Identifiers (OId's) where each OId uniquely identifies a transfer- or format syntax the origin supports. (**Note, that the list must not contain the OId's identifying ACSE and SR/Z39.50**)

To fill-in component "Presentation Context Definition List" of the AAssociate.Request service primitive function DbvAssociateRequest() creates a list which contains

- the ACSE OId

- the SR and/or Z39.50 OId (depending on the value(s) supplied for parameter "Application Context Name")

- the "Presentation Context Definition List" value(s) provided in the "ACSE-configuration-file".

Upon receiving the AAssociate.Confirmation service primitive, the function checks the component "Presentation Context Definition Result List" and determines, which of the abstract syntaxes proposed by the local application had been accepted by the remote application. Those record and format syntaxes which had been accepted by the remote application are assigned to the list which component *ppcCommonSyntaxes* of the *DbvProtocolAndSyntaxes* structure refers to. The list is terminated by a pointer (char*)NULL. **(Note, that the list does not contain the ACSE and SR/Z39.50 OId's)**. If the list is not empty, parameter *peAbstractSyntax* indicates "Common_Syntaxes".

If there is no value provided for "Presentation Context Definition Result List" in the service primitive or if there is no common record or format syntax, the function assigns the value "No_Syntaxes_Provided" respectively "No_Common_Syntax" to parameter *peAbstractSyntax*. Parameter *ppcCommonSyntaxes* is assigned the value (char**)NULL.

The behaviour of function DbvAssociateRequest() in case the ACSE-configuration-file is missing or disrupted, is described in Annex B.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget                *psTargetId = (DbvTarget*)NULL;
DbvAssociateState         eAssocState = Associate_Null;
DbvAssociateResult       *puAssociateResult = NULL;
DbvError                 *psError = NULL;


/* get buffer for and fill-in structure DbvTarget */
if (DbvAssociateRequest(&lAssocId, CALL_BLOCKING, psTargetId, &eAssocState,
                     &puAssociateResult, &psError) == SI_OK)
{
   if (eAssocState == Associate_Accepted)
   {
      /*
          Inspect puAssociateResult->psProtocolAndSyntaxes.
          If the values are not acceptable, call function DbvReleaseRequest().
      */
          FreeAssociateResult(Associate_Accepted, &puAssociateResult);
   }
   else if (eAssocState == Associate_Rejected)
   {
      /*
          Inspect puAssociateResult->psAssociateReject
          notify user */
      */
          FreeAssociateResult(Associate_Rejected, &puAssociateResult);
   }
   else
   {
       /* bug in function DbvAssociateRequest() */
   }
}
else
{
   PrivateErrorHandling(psError);
                 :
                 :
}

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

See Also:

DbvConfirmAssociateRequest(), DbvAbortRequest(), DbvAssociateResponse(), SISetACSEFile().

3.1.1.2 DbvReleaseRequest()

Purpose:

Release application association using the ACSE-service ARelease.Request.

Note: Calling this function is only permitted, if there are no outstanding application specific responses (e.g. SearchResponse, PresentResponse etc.).

Function Name and Parameters:

```
DbvReleaseRequest (
DbvAssocId          lAssocId,          /* IN */
DbvSecs             lSeconds,          /* IN */
DbvReleaseState    * peReleaseState,   /* OUT_C */
DbvError          **ppsSIError         /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association to be released.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately. If the function returns SI_OK, the value which parameter *peReleaseState* refers to, will most probably indicate ***Release_Pending***. Note however, that the value may also indicate *Release_Accepted* or *Release_Rejected*, in which case the association release already terminated. A value ***CALL_BLOCKING*** causes the function to wait for either the expected response or an event that indicates that any error occured. If the function returns SI_OK, the association was either released, or the release request was rejected by the remote user. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of the expected response (or any other event indicating a failure situation). If the function returns SI_OK, the value which parameter *peReleaseState* refers to indicates the state of the association. If the state is ***Release_Pending***, i.e. the response was not yet received, use either function DbvAbortRequest() (refer to 3.1.1.3) to abort the association or use function DbvConfirmReleaseRequest() (refer to 3.1.1.5).

peReleaseState	A pointer to a value of type *DbvReleaseState*. The value should be ***Release_Null*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value is either ***Release_Accepted*** or ***Release_Rejected*** or ***Release_Pending***. ***Release_Accepted*** indicates, that the association was released. ***Release_Rejected*** indicates that the association release request was rejected by the remote user. ***Release_Pending*** indicates, that no response arrived and no other event occured until now. This value may occur only if the actual value of lSeconds is ***CALL_ASYNCHRONOUS*** or a value greater than zero.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

Using the value "normal" as the "release-request-reason", the function forms an ARelease.Request service primitive. The service primitive is presented to the ACSE service provider. The ACSE service provider submits a RLRQ-APDU.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association still exists.

The meaning of a return value SI_OK depends on the value of lSeconds (see description above).

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.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 if (eReleaseState == Release_Rejected)
   {
      /*
           In the SR-/Z39.50 context, the target system is
           not allowed to reject the relaese request.
           Abort the association using function DbvAbortRequest().
      */
   }
   else
   {
       /* bug in function DbvReleaseRequest() */
   }
}
else
{
   PrivateErrorHandling(psError);
                 :
                 :
}

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

See Also:

DbvConfirmReleaseRequest(), DbvAbortRequest(), DbvReleaseResponse().

3.1.1.3 DbvAbortRequest()

Purpose:

Abort application association using the ACSE-service AAbort.Request.

Function Name and Parameters:

```
DbvAbortRequest (
DbvAssocId          LAssocId,        /* IN */
DbvError            **ppsSIError     /* OUT_F */
)
```

Return Value:

SI_OK: if no error occurred.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association to be aborted, as returned by a call of function DbvAssociateRequest() (DbvReceiveAssociateRequest() in case of the Target).
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function presents an empty AAbort.Request service primitive to the ACSE service provider. The ACSE service provider submits an ABRT-APDU.

If the function returns SI_OK the association was aborted.

If the function returns SI_NOTOK, an error occured and the association was possibly not aborted. Component *eAssociationState* of the error structure gives the exact answer.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.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 */
                 :
                 :
}
```

See Also:

DbvAssociateRequest(), DbvReleaseRequest().

3.1.1.4 DbvConfirmAssociateRequest()

Purpose:

Ask for the completion of a previously initiated association establishment. The association establishment was initiated by a call of function DbvAssociateRequest().

Function Name and Parameters:

```
DbvConfirmAssociateRequest (
DbvAssocId              lAssocId,                /* IN */
DbvSecs                 lSeconds,                /* IN */
DbvAssociateState       *peAssociateState,       /* OUT_C */
DbvAssociateResult      **ppuAssociateResult,    /* OUT_F */
DbvError                **ppsSIError             /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association to be established.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, i.e. the function does not wait for the response. If the function returns SI_OK, and the response was available, the value which parameter *peAssociateState* refers to, either indicates ***Associate_Accepted*** or ***Associate_Rejected***. If the function returns SI_OK and the response was not available, the value which parameter *peAAssociateState* refers to indicates ***Associate_Pending***. A value ***CALL_BLOCKING*** causes the function to wait for either the expected response or an event that indicates that the association could not be established. If the function returns SI_OK, the association was either established, or rejected by the remote user or the remote ACSE-provider. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of the expected response (or any other event indicating that the association could not be established). If the function returns SI_OK, the value which parameter *peAssociateState* refers to indicates the state of the association. If the state is ***Associate_Pending***, i.e. the response was not yet received, use either function DbvAbortRequest() (refer to 3.1.1.3) to abort the association establishment or use again function DbvConfirmAssociateRequest().

peAssociateState	A pointer to a value of type *AAssociateState*. The value is updated only if the function returns SI_OK. In this case the value is either ***Associate_Accepted*** or ***Associate_Rejected*** or still ***Associate_Pending***. ***Associate_Accepted*** indicates, that the association was established. ***Associate_Rejected*** indicates that the association was rejected by either the remote user or the remote ACSE-provider. ***Associate_Pending*** indicates, that no response arrived and no other event occured until now. This value may occur only if the actual value of *lSeconds* is ***CALL_ASYNCHRONOUS*** or a value greater than zero.
ppuAssociateResult	The address of a pointer of type **DbvAssociateResult*. When calling the function, the pointer should refer to (DbvAssociateResult*)NULL. The pointer is updated only if the function returns SI_OK and the value which parameter *peAssociateState* refers to either indicates ***Associate_Accepted*** or ***Associate_Rejected***. If the value which parameter *peAssociateState* refers to indicates ***Associate_Accepted***, the pointer refers to a *DbvProtocolAndSyntaxes* structure indicating the application protocol to be used over the association and the transfer syntaxes (e.g. UNIMARC, MAB) commonly supported by the origin and the target system. If the value which parameter *peAssociateState* refers to indicates ***Associate_Rejected***, the pointer refers to a *DbvAssociateReject* structure indicating the reason for the rejection.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is used to ask for the completion of a previously initiated association establishment.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association still exists.

The meaning of a return value SI_OK depends on the value of *lSeconds* (see description above).

See also notes in chapter 3.1.1.1.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget               *psTargetId = (DbvTarget*)NULL;
DbvAssociateState        eAssocState = Associate_Null;
DbvAssociateResult      *puAssociateResult = NULL;
DbvError                *psError = NULL;

#define MAX_WAIT_TIME  5L;


/* get buffer for and fill-in structure DbvTarget */
if (DbvAssociateRequest(&lAssocId, CALL_ASYNCHRONOUS, psTargetId, &eAssocState,
                     &puAssociateResult, &psError) == SI_OK)
{
   if (eAssocState == Associate_Pending)
   {
      /*
           do whatever you want but don't try to send messages
      */
      if (DbvConfirmAssociateRequest(lAssocId, MAX_WAIT_TIME, &eAssocState,
&puAssociateResult,
                                  &psError) == SI_OK)
      {
         if (eAssocState == Associate_Accepted)
         {
           /*
             Inspect puAssociateResult->psProtocolAndSyntaxes.
             If the values are not acceptable, call function
             DbvReleaseRequest().
           */
         }
         else if (eAssocState == Associate_Rejected)
         {
            /*
               notify user, free buffer
            */
         }
         else if (eAssocState == Associate_Pending)
         {
            /* e.g. */
            if (DbvAbortRequest(lAssocId, &psError) == SI_OK)
            {
                /*   notify user, free buffer */
            }
            else
            {
               PrivateErrorHandling(psError);
               FreeErrorCode(&psError);
            }
         }
         else
```

```
         {
            /* bug in function DbvConfirmAssociateRequest() */
         }
      }
      else
      {
         PrivateErrorHandling(psError);
         FreeErrorCode(&psError);
                 :
                 :
      }
   else if (eAssocState == Associate_Accepted)
   {
            :
            :
   }
   else if (eAssocState == Associate_Rejected)
   {
            :
            :
   }
   else
   {
       /* bug in function DbvAssociateRequest() */
   }
}
else
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
                 :
                 :
}
```

See Also:

DbvAssociateRequest(), DbvAssociateResponse(), DbvAbortRequest().

3.1.1.5 DbvConfirmReleaseRequest()

Purpose:

Ask for the completion of a previously initiated association release. The association release was initiated by a call of function DbvReleaseRequest().

Function Name and Parameters:

```
DbvConfirmReleaseRequest (
DbvAssocId             lAssocId,          /* IN */
DbvSecs                lSeconds,          /* IN */
DbvReleaseState        *peReleaseState,   /* OUT_C */
DbvError               **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association to be released.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, i.e. the function does not wait for the required response. If the function returns SI_OK, and the response was available, the value which parameter *peReleaseState* refers to, either indicates ***Release_Accepted*** or ***Release_Rejected***. If the function returns SI_OK and the response was not available, the value which parameter *peReleaseState* refers to indicates ***Release_Pending***. A value ***CALL_BLOCKING*** causes the function to wait for either the expected response or an event that indicates that any error occured. If the function returns SI_OK, the association was either released, or the release request was rejected by the remote user. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of the expected response (or any other event indicating a failure situation). If the function returns SI_OK, the value which parameter *peReleaseState* refers to indicates the state of the association. If the state is still ***Release_Pending***, i.e. the response was not yet received, use either function DbvAbortRequest() (refer to 3.1.1.3) to abort the association release or use again function DbvConfirmReleaseRequest().

peReleaseState	A pointer to a value of type *DbvReleaseState*. The value should be ***Release_Null*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value is either ***Release_Accepted*** or ***Release_Rejected*** or ***Release_Pending***. ***Release_Accepted*** indicates, that the association was released. ***Release_Rejected*** indicates that the release request was rejected by the remote user. In the SR-/Z39.50 context, the target system is not allowed to reject a release request. ***Release_Pending*** indicates, that no response arrived and no other event occured until now.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is used to ask for the completion of a previously initiated association release.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association still exists.

The meaning of a return value SI_OK depends on the value of *lSeconds* (see description above).

Usage:

According to 3.1.1.4

See Also:

DbvReleaseRequest(), DbvReleaseResponse(), DbvAbortRequest().

3.1.2 Association Handling Functions Using TCP-Services

Note:

When layering the SR-/Z39.50 service provider directly on top of TCP/IP, there is no real association establishment at the application layer. Rather there is a connection establishment on transport level. As soon as the transport connection is established, both applications, client and server, may start sending data.

The server application has no means to reject the connection establishment. If the server application is not willing to serve a particular client, it has to immediately close or abort the already existing transport (TCP) connection.

The following functions are available to the API-user:

- DbvAssociateRequest()
- DbvReleaseRequest()
- DbvAbortRequest()
- DbvConfirmAssociateRequest()
- DbvConfirmReleaseRequest()

Chapters 3.1.2.1 - 3.1.2.5 describe these functions.

3.1.2.1 DbvAssociateRequest()

Purpose:

Open TCP connection using the TCP service OPEN (active mode).

Note: Unlike when using the ACSE service AAssociate.Request, the remote application cannot reject the associate request and it is not possible to negotiate an "Application Context" and a "Presentation Context" between the two applications. This means:

- an association state *Associate_Rejected* may never occur
- if the function returns SI_OK, the value which parameter *peApplicationProtocol* (of the *DbvProtocolAndSyntaxes* structure) refers to, will always indicate *Unknown_Protocol*
- if the function returns SI_OK, the value which parameter *peAbstractSyntax* (of the *DbvProtocolAndSyntaxes* structure) refers to, will always indicate *No_Syntaxes_Provided*.

Function Name and Parameters:

```
DbvAssociateRequest (
DbvAssocId             *plAssocId,                /* OUT_C */
DbvSecs                 lSeconds,                 /* IN */
DbvTarget              *psTargetIdentification,   /* IN */
DbvAssociateState      *peAssociateState,         /* OUT_C */
DbvAssociateResult    **ppuAssociateResult,       /* OUT_F */
DbvError              **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

plAssocId

A pointer to a *long* value.
In case the function returns SI_OK and the connection was either accepted or the connect request is still pending, the value is updated and provides the unique identifier to be used to subsequently refer to the connection.
In case the function returns SI_NOTOK, the value is unchanged.

lSeconds

A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately. If the function returns SI_OK, then the value parameter *plAssocId* refers to, is the unique identifier which is to be used in subsequent calls to function DbvConfirmAssociateRequest() (refer to 3.1.2.4) or any other function provided to the API-user. The value, which parameter *peAssociateState* refers to, most probably will indicate ***Associate_Pending***. Note however, that the value may also indicate *Associate_Accepted*, in which case the connection establishment already terminated.
A value ***CALL_BLOCKING*** causes the function to wait for either the completion of the connection establishment or an event (e.g. timeout) indicating that the connection could not be established. If the function returns SI_OK, the connection was established.
Any other value greater than zero causes the function to wait the given number of seconds for the completion of the connection establishment (or any other event indicating that the connection could not be established). If the function returns SI_OK, the value parameter *peAssociateState* refers to indicates the state of the connection establishment. If the state is ***Associate_Pending***, i.e. the connection is not yet established, use either function DbvAbortRequest() (refer to 3.1.2.3) to abort the connection establishment or use function DbvConfirmAssociateRequest() (refer to 3.1.2.4).

psTargetIdentification	A pointer to a value of type *DbvTarget*. The value provides either the name or the address of the target system, the association has to be established to.
peAssociateState	A pointer to a value of type *DbvAssociateState*. The value should be ***Associate_Null*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value is either ***Associate_Accepted*** or ***Associate_Pending***. ***Associate_Accepted*** indicates, that the connection was established. ***Associate_Pending*** indicates, that the connection establishment was not completed and no other event occured until now. This value may occur only if the actual value of *lSeconds* is ***CALL_ASYNCHRONOUS*** or a value greater than zero.
ppuAssociateResult	The address of a pointer of type **DbvAssociateResult*. When calling the function, the pointer should refer to (DbvAssociateResult*)NULL. The pointer is updated only if the function returns SI_OK and the value which parameter *peAssociateState* refers to indicates ***Associate_Accepted***. In this case the pointer refers to a *DbvProtocolAndSyntaxes* structure where component *peApplicationProtocol* indicates ***Unknown_Protocol*** and component *peAbstractSyntax* indicates ***No_Syntaxes_Provided***.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psTargetIdentification* the name or the address of the target system the connection has to be established to.

The function determines the address of the target system and opens a TCP connection using the TCP service OPEN in active mode.

If the function returns SI_NOTOK, an error occured and the connection was most probably not established. Component *eAssociationState* of the error structure gives the exact answer.

The meaning of a return value SI_OK depends on the actual value of *lSeconds* (see description above).

If the connection was established or the connect request is still pending, the value which parameter *plAssocId* refers to is the unique identifier of the association. This identifier is to be used in all subsequent function calls to any association handling and/or SR-/Z39.50 function.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget               *psTargetId = (DbvTarget*)NULL;
DbvAssociateState        eAssocState = Associate_Null;
DbvAssociateResult      *puAssociateResult = NULL;
DbvError                *psError = NULL;


/* get buffer for and fill-in structure DbvTarget */
if (DbvAssociateRequest(&lAssocId, CALL_BLOCKING, psTargetId, &eAssocState,
    &puAAssociateResult, &psError) == SI_OK)
{
   if (eAssocState == Associate_Accepted)
   {
      /*
          Inspect puAssociateResult->psProtocolAndSyntaxes
      */
          FreeAssociateResult(Associate_Accepted, &puAssociateResult);
   }
   else
   {
       /* bug in function DbvAssociateRequest() */
   }
}
else
{
   PrivateErrorHandling(psError);
                 :
                 :
}

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

See Also:

DbvConfirmAssociateRequest(), DbvAssociateResponse(), DbvAbortRequest().

3.1.2.2 DbvReleaseRequest()

Purpose:

Release transport connection using the TCP service CLOSE.

Note: Calling this function is only permitted, if there are no outstanding application specific responses (e.g. SearchResponse, PresentResponse etc.).

Function Name and Parameters:

```
DbvReleaseRequest (
DbvAssocId          lAssocId,           /* IN */
DbvSecs             lSeconds,           /* IN */
DbvReleaseState    * peReleaseState,    /* OUT_C */
DbvError          **ppsSIError          /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the connection to be closed.
lSeconds	In the TCP/IP context, this parameter has no meaning, i.e. in case the function call is permitted, the function immediately returns and indicates ***Release_Accepted***.
peReleaseState	A pointer to a value of type *DbvReleaseState*. The value should be ***Release_Null*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value indicates ***Release_Accepted***.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function invokes the TCP service CLOSE to indicate to the target application that the origin will not send any more data.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the connection still exists.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.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 */
                 :
}
```

See Also:

DbvConfirmReleaseRequest(), DbvReleaseResponse(), DbvAbortRequest().

3.1.2.3 DbvAbortRequest()

Purpose:

Abort the transport connection using the TCP service CLOSE.

Function Name and Parameters:

```
DbvAbortRequest (
DbvAssocId          LAssocId,           /* IN */
DbvError            **ppsSIError        /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the connection to be aborted, as returned by a call of function DbvAssociateRequest() (DbvReceiveAssociateRequest() in case of the Target).
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function invokes the TCP service CLOSE.

If the function returns SI_OK the connection was aborted.

If the function returns SI_NOTOK, an error occured and the connection was possibly not aborted. Component *eAssociationState* of the error structure gives the exact answer.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.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 */
                 :
                 :
}
```

See Also:

DbvAssociateRequest(), DbvReleaseRequest().

3.1.2.4 DbvConfirmAssociateRequest()

Purpose:

Ask for the completion of a previously initiated connection establishment. The connection establishment was initiated by a call of function DbvAssociateRequest().

Function Name and Parameters:

```
DbvConfirmAssociateRequest (
DbvAssocId              lAssocId,               /* IN */
DbvSecs                 lSeconds,               /* IN */
DbvAssociateState       *peAssociateState,      /* OUT_C */
DbvAssociateResult      **ppuAssociateResult,   /* OUT_F */
DbvError                **ppsSIError            /* OUT_F */
)
```

For the description refer to 3.1.2.1 and 3.1.1.4.

3.1.2.5 DbvConfirmReleaseRequest()

There is no need for using this function in a TCP/IP environment (refer to 3.1.2.2). The function will always return SI_NOTOK.

3.1.3 SR-/Z39.50 Functions

Chapters 3.1.3.1 - 3.1.3.13 describe the functions available for sending SR-/Z39.50 data.

Chapter 3.1.3.14 describes the function provided for receiving data.

In both environments, the OSI environment and the TCP environment, the functions used for sending data encode the data to be transferred according to the "Basic Encoding Rules" (BER).

In the OSI environment the functions use the Presentation services for sending and receiving data.

In the TCP environment the functions use TCP services for sending and receiving data.

3.1.3.1 DbvInitializeRequest()

Purpose:

Send InitializeRequest-APDU.

Function Name and Parameters:

```
DbvInitializeRequest (
DbvAssocId                  lAssocId,           /* IN */
DbvInitializeReq           * psInitializeReq,   /* IN */
DbvError                  **ppsSIError          /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psInitializeReq	Pointer to the data to be transferred. The definition of the data type *DbvInitializeReq* corresponds to the definition of the INITIALIZE.Request service primitive.

ppsSIError — The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL.
The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psInitializeReq* the data corresponding to an INITIALIZE.Request service primitive. The function checks the data for completeness, forms and submits an InitializeRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.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)
{
   SIFreeInitializeReq(&psInitializeReq);
                 :
                 :
}
else
{
   PrivateErrorHandling(psError);
   SIFreeInitializeReq(&psInitializeReq);
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.1.3.2 DbvSearchRequest()

Purpose:

Send SearchRequest-APDU.

Function Name and Parameters:

```
DbvSearchRequest (
DbvAssocId          lAssocId,        /* IN */
DbvSearchReq      * psSearchReq,     /* IN */
DbvError         **ppsSIError        /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psSearchReq	Pointer to the data to be transferred. The definition of the data type *DbvSearchReq* corresponds to the definition of the SEARCH.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psSearchReq* the data corresponding to a SEARCH.Request service primitive. The function checks the data for completeness, forms and submits a SearchRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.3 DbvPresentRequest()

Purpose:

Send PresentRequest-APDU.

Function Name and Parameters:

```
DbvPresentRequest (
DbvAssocId             lAssocId,          /* IN */
DbvPresentReq         * psPresentReq,     /* IN */
DbvError             **ppsSIError         /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psPresentReq	Pointer to the data to be transferred. The definition of the data type *DbvPresentReq* corresponds to the definition of the PRESENT.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psPresentReq* the data corresponding to a PRESENT.Request service primitive. The function checks the data for completeness, forms and submits a PresentRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.4 DbvDeleteResultSetRequest()

Purpose:

Send DeleteResultSetRequest-APDU.

Function Name and Parameters:

```
DbvDeleteResultSetRequest (
DbvAssocId                 lAssocId,                /* IN */
DbvDeleteResultSetReq     * psDeleteResultSetReq,   /* IN */
DbvError                 **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psDeleteResultSetReq	Pointer to the data to be transferred. The definition of the data type *DbvDeleteResultSetReq* corresponds to the definition of the DELETE-RESULT-SET.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psDeleteResultSetReq* the data corresponding to a DELETE-RESULT-SET.Request service primitive. The function checks the data for completeness, forms and submits a DeleteResultSetRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.5 DbvScanRequest()

Purpose:

Send ScanRequest-APDU.

Used by: Origin-System.

Function Name and Parameters:

```
DbvScanRequest (
DbvAssocId        lAssocId,       /* IN */
DbvScanReq        * psScanReq,    /* IN */
DbvError          **ppsSIError    /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psScanReq	Pointer to the data to be transferred. The definition of the data type *DbvScanReq* corresponds to the definition of the SCAN.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psScanReq* the data corresponding to a SCAN.Request service primitive. The function checks the data for completeness, forms and submits a ScanRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.6 DbvResourceReportRequest()

Purpose:

Send ResourceReportRequest-APDU.

Function Name and Parameters:

```
DbvResourceReportRequest (
DbvAssocId                    lAssocId,                  /* IN */
DbvResourceReportReq          * psResourceReportReq,     /* IN */
DbvError                      **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psResourceReportReq	Pointer to the data to be transferred. The definition of the data type *DbvResourceReportReq* corresponds to the definition of the RESOURCE-REPORT.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psResourceReportReq* the data corresponding to a RESOURCE-REPORT.Request service primitive. The function checks the data for completeness, forms and submits a ResourceReportRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.7 DbvSortRequest()

Purpose:

Send SortRequest-APDU.

Function Name and Parameters:

```
DbvSortRequest (
DbvAssocId          lAssocId,        /* IN */
DbvSortReq         * psSortReq,      /* IN */
DbvError          **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psSortReq	Pointer to the data to be transferred. The definition of the data type *DbvSortReq* corresponds to the definition of the SORT.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psSortReq* the data corresponding to a SORT.Request service primitive. The function checks the data for completeness, forms and submits a SortRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.8 DbvExtendedServicesRequest()

Purpose:

Send ExtendedServicesRequest-APDU.

Function Name and Parameters:

```
DbvExtendedServicesRequest (
DbvAssocId                      lAssocId,                  /* IN */
DbvExtendedServicesReq          * psExtendedServicesReq,   /* IN */
DbvError                        **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
PsExtendedServicesReq	Pointer to the data to be transferred. The definition of the data type *DbvExtendedServicesReq* corresponds to the definition of the EXTENDED-SERVICES.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psExtendedServicesReq* the data corresponding to an EXTENDED-SERVICES.Request service primitive. The function checks the data for completeness, forms and submits an ExtendedServicesRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.9 DbvCloseRequest()

Purpose:

Send CloseRequest-APDU.

Function Name and Parameters:

```
DbvCloseRequest (
DbvAssocId        lAssocId,       /* IN */
DbvCloseReq     * psCloseReq,     /* IN */
DbvError        **ppsSIError      /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psCloseReq	Pointer to the data to be transferred. The definition of the data type *DbvCloseReq* corresponds to the definition of the CLOSE.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psCloseReq* the data corresponding to a CLOSE.Request service primitive. The function checks the data for completeness, forms and submits a CloseRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.10 DbvCloseResponse()

Purpose:

Send CloseResponse-APDU.

Function Name and Parameters:

```
DbvCloseResponse (
DbvAssocId            lAssocId,      /* IN */
DbvCloseReq           * psCloseRsp,  /* IN; there is only one APDU defined for CLOSE */
DbvError              **ppsSIError   /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psCloseRsp	Pointer to the data to be transferred. The definition of the data type *DbvCloseReq* corresponds to the definition of the CLOSE.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psCloseRsp* the data corresponding to a CLOSE.Response service primitive. The function checks the data for completeness, forms and submits a CloseResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.3.1.

3.1.3.11 DbvAccessControlResponse()

Purpose:

Send Access Control Response APDU.

Function Name and Parameters:

```
DbvAccessControlResponse (
DbvAssocId                     lAssocId,               /* IN */
DbvAccessControlRsp           * psAccessControlRsp,    /* IN */
DbvError                     **ppsSIError              /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psAccessControlRsp	Pointer to the data to be transferred. The definition of the data type *DbvAccessControlRsp* corresponds to the definition of the ACCESS CONTROL Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psAccessControlRsp* the data corresponding to an ACCESS CONTROL Response service primitive. The function checks the data for completeness, forms and submits an AccessControlResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1.

3.1.3.12 DbvResourceControlResponse()

Purpose:

Send Resource Control Response APDU.

Function Name and Parameters:

```
DbvResourceControlResponse (
DbvAssocId                 lAssocId,                  /* IN */
DbvResourceControlRsp     * psResourceControlRsp,     /* IN */
DbvError                 **ppsSIError                 /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psResourceControlRsp	Pointer to the data to be transferred. The definition of the data type *DbvResourceControlRsp* corresponds to the definition of the RESOURCE CONTROL Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psResourceControlRsp* the data corresponding to an RESOURCE CONTROL Response service primitive. The function checks the data for completeness, forms and submits an ResourceControlResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1.

3.1.3.13 DbvTriggerResourceControlRequest()

Purpose:

Send Trigger Resource Control Request APDU.

Function Name and Parameters:

DbvTriggerResourceControlRequest (

DbvAssocId	LAssocId,	/* IN */
DbvTriggerResourceControlReq	* psTriggerResourceControlReq,	/* IN */
DbvError	**ppsSIError	/* OUT_F */

)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psTriggerResourceControlRsp	Pointer to the data to be transferred. The definition of the data type *DbvTriggerResourceControlReq* corresponds to the definition of the TRIGGER RESOURCE CONTROL Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psResourceControlRsp* the data corresponding to a TRIGGER RESOURCE CONTROL Request service primitive. The function checks the data for completeness, forms and submits a TriggerResourceControlRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.1.3.14 DbvReceiveDataOrigin()

Purpose:

Receive any SR-/Z39.50 data.

Used by: Origin-System.

Function Name and Parameters:

```
DbvReceiveDataOrigin (
DbvAssocId          lAssocId,           /* IN */
DbvSecs             lSeconds,           /* IN */
DbvOriginData     ** ppsOriginData,     /* OUT_F */
DbvError          **ppsSIError          /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, i.e. if there are currently no data available, the function does not wait for any incoming data. If the function returns SI_OK, then the pointer associated with parameter *ppsOriginData* either was not updated, in which case no data had been available, or refers to a filled-in *DbvOriginData* structure providing the received data. A value ***CALL_BLOCKING*** causes the function to wait for incoming data (or any event indicating a failure). If the function returns SI_OK, the pointer associated with parameter *ppsOriginData* refers to the received data. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of any data (or any other event indicating a failure). If the function returns SI_OK, then the pointer associated with *ppsOriginData* either was not updated, in which case no data were available, or refers to a filled-in *DbvOriginData* structure providing the received data.

ppsOriginData	The address of a pointer of type **DbvOriginData*. When calling the function, the pointer should refer to (DbvOriginData*)NULL. The pointer is updated only if the function returns SI_OK. In this case the pointer refers to a filled-in *DbvOriginData* structure.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is used to receive SR-/Z39.50 responses, submitted by the "Target System".

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association still exists.

The meaning of a return value SI_OK depends on the value of *lSeconds* (see description above).

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


if (DbvReceiveDataOrigin(lAssocId, CALL_BLOCKING, &psOriginData, &psError) ==
SI_OK)
{
   /* process data */
   SIFreeDbvOriginData(&psOriginData);
       :
       :
}
else
{
  PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
}
```

See also:

Functions described in chapters 3.2.4.1 - 3.2.4.12.

3.2 Functions Used by the Target System

Chapter 3.2.1 describes the Target initialization function (DbvTargetInitialize()) which has to be called prior to any other function. (This function must not be confused with the SR/Z39.50 Initialize function)

Chapters 3.2.2 and 3.2.3 describe the association handling functions available to the API-user in the OSI-, and TCP-environment respectively.

Chapter 3.2.4 describes the functions avaiable for sending and receiving SR-/Z39.50 data in both environments.

3.2.1 DbvTargetInitialize()

Purpose:

Initialize the target process.

Function Name and Parameters:

DbvTargetInitialize (

```
int          argc,            /* IN */
char         **argv,          /* IN */
DbvError     **ppsSIError     /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

argc	argc parameter of the target main program. If using TCP stack software, the expected value of argc is 3. If using OSI stack, argc is provided by the TSAPD OSI stack daemon.

argv	argv parameter of the target program main(). If using TCP stack: - First argument is the name of the executable of the target program, - Second argument is the string "TCP" for a TCP/IP stack - Second argument is the string "TCP" for a TCP/IP stack - Third argument is the file descriptor of the socket to which the Origin is connected. If using OSI stack, parameters are private between the TSAPD daemon and the ISODE libraries.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Note that the TCP/IP Target daemon program called targetd_tcpip provides argc and argv values that can be passed direct to this function.

3.2.2 Association Handling Functions Using the ACSE-Services

The following functions are available to the API-user:

- DbvReceiveAssociateRequest()
- DbvAssociateResponse()
- DbvReleaseResponse()
- DbvAbortRequest()

Chapters 3.2.2.1 - 3.2.2.4 describe these functions.

3.2.2.1 DbvReceiveAssociateRequest()

Purpose:

Receive associate request using the ACSE-services.

This is the first communication related function the target process has to call.

Function Name and Parameters:

```
DbvReceiveAssociateRequest (
DbvAssocId                   *plAssocId,                /* OUT_C */
DbvSecs                        lSeconds,                /* IN */
char                         **ppcOriginAddress,        /* OUT_F */
DbvProtocolAndSyntaxes       **ppsProtocolAndSyntaxes,  /* OUT_F */
DbvError                     **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

plAssocId	A pointer to a long value. In case the function returns SI_OK and an associate request was received, the value is updated and provides the unique identifier of the association to be established.

lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, if there is currently no associate request from any remote system. If the function returns SI_OK and there was an associate request from any remote system, then the pointer associated with parameter *ppcOriginAddress* and the pointer associated with parameter *ppsProtocolAndSyntaxes* is updated. The pointer associated with parameter *ppcOriginAddress* refers to a character string indicating the address of the origin system that requested the association. The pointer associated with parameter *ppsProtocolAndSyntaxes* refers to a filled-in structure *DbvProtocolAndSyntaxes* which indicates the protocol to be used over the association and the record and format syntaxes commonly supported by the origin and target application. If the function returns SI_OK, and there was no associate request the above pointers are not updated. A value ***CALL_BLOCKING*** causes the function to wait for the incoming of an associate request. If the function returns SI_OK, the pointer associated with parameter *ppcOriginAddress* and the pointer associated with parameter *ppsProtocolAndSyntaxes* are updated as described above. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of an associate request (or any other event indicating a failure situation). If the function returns SI_OK, and there was an associate request from any remote system, the pointer associated with parameter *ppcOriginAddress* and the pointer associated with parameter *ppsProtocolAndSyntaxes* are updated as described above. If the function returns SI_OK, and there was no associate request from any remote system, then the pointers are not updated.
ppcOriginAddress	The address of a pointer of type **char*. When calling the function, the pointer should refer to (char*)NULL. The pointer is updated only if the function returns SI_OK and any remote system requested an association. In this case the pointer refers to a character string indicating the address of the system requesting the association.
ppsProtocolAndSyntaxes	The address of a pointer of type **DbvProtocolAndSyntaxes*. The pointer is updated only if the function returns SI_OK and a Transport (more exactly Presentation) connection was established. In this case the pointer refers to a *DbvProtocolAndSyntaxes* structure which indicates the protocol to be used over the association and the abstract syntaxes commonly supported by both applications.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is called to serve an associate request from any remote system.

The meaning of a return value SI_OK depends on the actual value of *lSeconds*.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association exists.

Upon receiving an AAssociate.Indication service primitive, the function checks the values of the parameters "Application Context Name" and "Presentation Context Definition List" provided in the service primitive.

The permitted application contexts (currently "Basic_SR_Application_Context" and "Basic_Z3950_Appliaction_Context") are known to the function. The application context(s) and the record- and format syntaxes supported by the target have to be provided in the targets "ACSE-configuration-file".

If the "Application Context Name" value provided in the service primitive is not a permitted value, component *peApplicationProtocol* of the *DbvProtocolAndSyntaxes* structure is set to indicate ***Unidentified_Protocol***. In this case the application on top of the API shall reject the associate request.

If the value provided in the service primitive is a permitted value, then the function determines the value to be assigned to component *peApplicationProtocol* of the structure as follows:

If the application on top of the API supports both, SR and Z39.50, the value is set according to the application context provided in the service primitive.

If the application supports only one protocol, then the value indicates the protocol supported by the application, regardless of the value provided in the service primitive. In this case, the value of "Application Context Name" in the AAssociate.Response service primitive (refer to function DbvAssociateResponse()) will indicate the application context supported by the application.

If there was a value provided for parameter "Presentation Context Definition List" in the AAssociate.Indication service primitive, the function checks the abstract syntaxes proposed by the originator against the values provided in the targets "ACSE-configuration-file". The function assignes to component *ppcCommonSyntaxes* of the *DbvProtocolAndSyntaxes* strúcture those abstract syntaxes , which are proposed by the origin system and supported by the local application **(Note, that the list does not contain the ACSE and SR/Z39.50 OId)**. If there are common syntaxes, component *peAbstractSyntax* of the *DbvProtocolAndSyntaxes* structure is assigned the value ***Common_Syntaxes***, otherwise the value ***No_Common_Syntaxes*** is assigned to that structure component.

If there was no value provided for parameter "Presentation Context Definition List" in the AAssociate.Indication service primitive, then the value ***No_Syntaxes_Provided*** is assigned to component p*eAbstractSyntax* of the *DbvProtocolAndSyntaxes* structure. In this case component *ppcCommonSyntaxes* will refer to (char**)NULL.

Inspecting the address of the origin system requesting the association and the protocol to be used over the association, the caller of the function has to decide whether or not to accept the associaton. In either case the caller of the function has to respond to the associate request by calling function *DbvAssociateResponse()*. In case of a fatal error situation a call of function *DbvAbortRequest()* is also permitted.

The behaviour of function DbvReceiveAssociateRequest() in case the ACSE-configuration-file is missing or disrupted, is described in Annex B.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                     lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
char                          *pcOriginAddress = NULL;
DbvApplicationProtocol         eApplicationProtocol = Unidentified_Protocol;
DbvError                      *psError = NULL;


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 */
```

See Also:

DbvAssociateRequest(), DbvAssociateResponse(), DbvAbortRequest().

3.2.2.2 DbvAssociateResponse()

Purpose:

Respond to an associate request using the ACSE-service AAssociate.Response.

Function Name and Parameters:

```
DbvAssociateResponse (
DbvAssocId              lAssocId,         /* IN */
DbvAssociateRsp         eAssociateRsp,    /* IN */
DbvError              **ppsSIError        /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association to be established, as returned by a call of function DbvReceiveAssociateRequest().
eAssociateRsp	One of ***Accepted***, ***Rejected_Permanent***, ***Rejected_Transient***. If the function returns SI_OK and the caller of the function accepted the association, then the association has been established. Otherwise the association has not been established.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *eAssociateRsp* the response to a previously received associate request. The caller of the function had either accepted or rejected the associate request.

The function forms an AAssociate.Response service primitive (see note below) and presents the service primitive to the ACSE-service provider. The ACSE-service provider submits an AARE-APDU.

If the function returns SI_OK and the caller of the function accepted the association, then the association is established.

If the function returns SI_OK and the caller of the function rejected the association, the association is not established.

If the function returns SI_NOTOK, an error occured and the association was most probably not established. Component *eAssociationState* of the error structure gives the exact answer.

Note:

The value which is assigned to parameter "Application Context Name" of the AAssociate.Response service primitive is the value that was previously determined by function DbvReceiveAssociateRequest().

The value which is assigned to component "Presentation Context Definition Result List" is determined as follows:

If there was no value provided for parameter "Presentation Context Definition List" in the AAssociate.Indication service primitive, then no value is assigned to the parameter.

If there was a value provided, the function checks the abstract syntaxes proposed by the originator against the values provided in the targets "ACSE-configuration-file". The function "accepts" those abstract syntaxes , which are proposed by the origin system and supported by the local application. Those abstract syntaxes which are not supported by the local application are marked as "rejected". The ACSE abstract syntax and the abstract syntax which corresponds to the "Application Context Name" are marked as "accepted".

The behaviour of function DbvAssociateResponse() in case the ACSE-configuration-file is missing or disrupted, is described in Annex B.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvAssociateRsp           eAssocRsp = Accepted;
DbvError                 *psError = NULL;


if (DbvAssociateResponse(lAssocId, eAssocRsp, &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 occured!!
   */

   PrivateErrorHandling(psError);
```

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

                 :
                 :
```

See Also:

DbvReceiveAssociateRequest(), DbvAssociateRequest(), DbvAbortRequest().

3.2.2.3 DbvReleaseResponse()

Purpose:

Respond to a release request using the ACSE-service ARelease.Response.

Function Name and Parameters:

```
DbvReleaseResponse (
DbvAssocId          lAssocId,           /* IN */
DbvReleaseRsp       *psAReleaseRsp,     /* IN */
DbvError            **ppsSIError        /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association to be released, as returned by a call of function DbvReceiveAssociateRequest().
psReleaseRsp	This parameter provides the response to a previously received release request (an ARelease.Indication service primitive). Since rejecting a release request is not permitted in the context of SR and Z39.50, the caller of the function shall provide the value (DbvReleaseRsp*) NULL. If the function returns SI_OK, the association was released.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psAReleaseRsp* the response to a release request.

Since the rejection of a release request is not permitted in the context of the SR and Z39.50 protocol, the parameter may refer to (DbvReleaseRsp*) NULL. In this case the function forms an ARelease.Response service primitive with parameter "result" indicating *affirmative* and parameter "reason" indicating *normal*.

In case the caller of the function tries to reject the release request, the function does not create a service primitive, returns SI_NOTOK and indicates to that rejecting the release request is not allowed.

If the release request was accepted, the function presents the service primitive to the ACSE service provider. The ACSE service provider submits a RLRE-APDU.

If the function returns SI_OK, the association was released.

If the function returns SI_NOTOK, an error occured. Component *eAssociationState* of the error structure indicates whether or not the association still exists.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                  lAssocId;
DbvReleaseRsp              *psReleaseRsp = NULL;
DbvError                   *psError = NULL;



if (DbvReleaseResponse(lAssocId, psReleaseRsp, &psError) == SI_OK)
{
   /* association released */
                  :
                  :
}
else
{
   /* if psError->eAssociationState == Association_Open
      the association still exists
   */

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

3.2.2.4 DbvAbortRequest()

Refer to 3.1.1.3.

3.2.3 Association Handling Functions Using TCP-Services

The following functions are available to the API-user:

- DbvReceiveAssociateRequest()
- DbvAssociateResponse()
- DbvReleaseResponse()
- DbvAbortRequest()

Chapters 3.2.3.1 - 3.2.3.4 describe these functions.

3.2.3.1 DbvReceiveAssociateRequest()

Purpose:

Receive the unique identifier of a transport connection and the address of the remote system that opened the connection.

This is the first communication related function the target process has to call.

Note: Unlike when using the ACSE services, the components *peApplicationProtocol* and *peAbstractSyntax* of the *DbvProtocolAndSyntaxes* structure will always indicate ***Unknown_Protocol*** and ***NO_Syntaxes_Provided.*** Component *ppcCommonSyntaxes* will always refer to (char**)NULL.

Function Name and Parameters:

```
DbvReceiveAssociateRequest (
DbvAssocId                *plAssocId,                 /* OUT_C */
DbvSecs                    lSeconds,                  /* IN */
char                      **ppcOriginAddress,         /* OUT_F */
DbvProtocolAndSyntaxes    **ppsProtocolAndSyntaxes,   /* OUT_F */
DbvError                  **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

plAssocId	A pointer to a *long* value. In case the function returns SI_OK and a transport connection exists, the value is updated and provides the unique identifier of the transport connection.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, if there was no transport connection established. If the function returns SI_OK and there was a transport connection established, then - the pointer associated with parameter *ppcOriginAddress* is updated and refers to a character string indicating the address of the origin system that requested the connection. - the pointer associated with parameter *ppsProtocolAndSyntaxes* is updated and refers to a filled in structure *DbvProtocolAndSyntaxes* (as described below). If the function returns SI_OK, and there was no connection established, the above pointers are not updated. A value ***CALL_BLOCKING*** causes the function to wait for the incoming of a transport connect request. If the function returns SI_OK, the pointers associated with parameters *ppcOriginAddress* and *ppsProtocolAndSyntaxes* are updated as described above. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of a transport connect request (or any other event indicating a failure situation). If the function returns SI_OK, the pointers associated with parameters *ppcOriginAddress* and *ppsProtocolAndSyntaxes* are updated as described above. If the function returns SI_OK, and there was no connection established, the above pointers are not updated.
ppcOriginAddress	The address of a pointer of type **char*. When calling the function, the pointer should refer to (char*)NULL. The pointer is updated only if the function returns SI_OK and a transport connection was established. In this case the pointer refers to a character string indicating the address of the system requesting the connection.
ppsProtocolAndSyntaxes	The address of a pointer of type **DbvProtocolAndSyntaxes*. The pointer is updated only if the function returns SI_OK and a transport connection was established. In this case the pointer refers to a *DbvProtocolAndSyntaxes* structure which indicates ***Unknown_Protocol*** and ***No_Syntaxes_Provided*** in the respective components.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is called to receive the unique identifier of a previously established transport connection and the address of the system that initiated the connection.

The meaning of a return value SI_OK depends on the actual value of *lSeconds*.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the connection exists.

Inspecting the address of the origin system that opened the transport connection, the caller of the function has to decide whether or not to serve the origin system. In either case the caller of the function has to respond by calling function *DbvAssociateResponse()*.

Usage: Refer to 3.2.2.1.

3.2.3.2 DbvAssociateResponse()

Purpose:

Decide whether or not to serve the requests of a particular origin system. The address of the origin system was received by a call of function DbvReceiveAssociateRequest().

Function Name and Parameters:

```
DbvAssociateResponse (
DbvAssocId            lAssocId,        /* IN */
DbvAssociateRsp       eAssociateRsp,   /* IN */
DbvError            **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the transport connection, as received by a call of function DbvReceiveAssociateRequest().
eAssociateRsp	One of ***Accepted***, ***Rejected_Permanent***, ***Rejected_Transient***. If the function returns SI_OK and the caller of the function provided one of the values ***Reject_Permanent*** or ***Reject_Transient***, the transport connection will be aborted.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

If parameter *eAssociateRsp* indicates ***Accepted***, the function returns immediately without performing any action.

If parameter *eAssociateRsp* indicates ***Reject_Permanent*** or Reject_Transient, the function aborts the transport connection by invoking the TCP service CLOSE.

If the function returns SI_NOTOK, an error occured. The error is reported in the *DbvError* structure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvAssociateRsp           eAssocRsp = Accepted;
DbvError                 *psError = NULL;


if (DbvAssociateResponse(lAssocId, eAssocRsp, &psError) == SI_OK)
{

   /* e.g. wait for incoming evets */
}
else
{
   /* if psError->eAssociationState == Association_Open
      the transport connection is still open even though an
      error occured!!
   */

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

                 :
                 :
```

3.2.3.3 DbvReleaseResponse()

Purpose:

Release transport connection using the TCP service CLOSE.

Function Name and Parameters:

```
DbvReleaseResponse (
DbvAssocId          lAssocId,          /* IN */
DbvReleaseRsp       *psAReleaseRsp,    /* IN */
DbvError            **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association to be released, as returned by a call of function DbvReceiveAssociateRequest().
psReleaseRsp	This parameter provides the response to a previously received release request (an ARelease.Indication service primitive). Since rejecting a release request is not permitted in the context of SR and Z39.50 (and is not possible at all when using the TCP service CLOSE), the caller of the function shall provide the value (DbvReleaseRsp*) NULL. If the function returns SI_OK, the association was released.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function invokes the TCP service CLOSE to indicate to the origin application that the target will not send any more data.

Since the TCP service CLOSE does not allow for any user data to be transferred, parameter *psReleaseRsp* may refer to (DbvReleaseRsp*) NULL. The function ignores this parameter.

If the function returns SI_OK, the transport connection was released.

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the connection still exists.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId              lAssocId;
DbvReleaseRsp          *psReleaseRsp = NULL;
DbvError               *psError = NULL;



if (DbvReleaseResponse(lAssocId, psReleaseRsp, &psError) == SI_OK)
{
   /* transport connection released */
                  :
                  :
}
else
{
   /* if psError->eAssociationState == Association_Open
      the connection still exists
   */

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

3.2.3.4 DbvAbortRequest()

Refer to 3.1.2.3.

3.2.4 SR-/Z39.50 Functions

Chapters 3.2.4.1 - 3.2.4.13 describe the functions available for sending SR-/Z39.50 data.

Chapter 3.2.4.14 describes the function provided for receiving data.

In both environments, the OSI environment and the TCP environment, the functions used for sending data encode the data to be transferred according to the "Basic Encoding Rules" (BER).

In the OSI environment the functions use the Presentation services for sending and receiving data.

In the TCP environment the functions use TCP services for sending and receiving data.

3.2.4.1 DbvInitializeResponse()

Purpose:

Send InitializeResponse-APDU.

Function Name and Parameters:

```
DbvInitializeResponse (
DbvAssocId              lAssocId,              /* IN */
DbvInitializeRsp       *psInitializeRsp,       /* IN */
DbvError              **ppsSIError             /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psInitializeRsp	Pointer to the data to be transferred. The definition of the data type *DbvInitializeRsp* corresponds to the definition of the INITIALIZE.Response service primitive.

ppsSIError The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL.
The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psInitializeRsp* the data corresponding to an INITIALIZE.Response service primitive. The function checks the data for completeness, forms and submits an InitializeResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvInitializeRsp        *psInitializeRsp = NULL;
DbvError                *psError = NULL;


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

    Allocate buffer for psInitializeRsp and fill in structure.
*/


if (DbvInitializeResponse(lAssocId, psInitializeRsp, &psError) == SI_OK)
{
   SIFreeInitializeRsp(&psInitializeRsp);
            :
            :
}
else
{
   PrivateErrorHandling(psError);
   SIFreeInitializeRsp(&psInitializeRsp);
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
               :
```

3.2.4.2 DbvSearchResponse()

Purpose:

Send SearchResponse-APDU.

Used by: Target-System.

Function Name and Parameters:

```
DbvSearchResponse (
DbvAssocId           lAssocId,          /* IN */
DbvSearchRsp        * psSearchRsp,      /* IN */
DbvError           **ppsSIError         /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psSearchRsp	Pointer to the data to be transferred. The definition of the data type *DbvSearchRsp* corresponds to the definition of the SEARCH.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psSearchRsp* the data corresponding to a SEARCH.Response service primitive. The function checks the data for completeness, forms and submits a SearchResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.3 DbvPresentResponse()

Purpose:

Send PresentResponse-APDU.

Function Name and Parameters:

```
DbvPresentResponse (
DbvAssocId          lAssocId,          /* IN */
DbvPresentRsp      * psPresentRsp,     /* IN */
DbvError          **ppsSIError         /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psPresentRsp	Pointer to the data to be transferred. The definition of the data type *DbvPresentRsp* corresponds to the definition of the PRESENT.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psPresentRsp* the data corresponding to a PRESENT.Response service primitive. The function checks the data for completeness, forms and submits a PresentResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.4 DbvDeleteResultSetResponse()

Purpose:

Send DeleteResultSetResponse-APDU.

Function Name and Parameters:

```
DbvDeleteResultSetResponse (
DbvAssocId                    lAssocId,                 /* IN */
DbvDeleteResultSetRsp        * psDeleteResultSetRsp,    /* IN */
DbvError                    **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psDeleteResultSetRsp	Pointer to the data to be transferred. The definition of the data type *DbvDeleteResultSetRsp* corresponds to the definition of the DELETE-RESULT-SET.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psDeleteResultSetRsp* the data corresponding to a DELETE-RESULT-SET.Response service primitive. The function checks the data for completeness, forms and submits a DeleteResultSetResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.5 DbvScanResponse()

Purpose:

Send ScanResponse-APDU.

Function Name and Parameters:

```
DbvScanResponse (
DbvAssocId          lAssocId,       /* IN */
DbvScanRsp         * psScanRsp,     /* IN */
DbvError          **ppsSIError      /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psScanRsp	Pointer to the data to be transferred. The definition of the data type *DbvScanRsp* corresponds to the definition of the SCAN.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psScanRsp* the data corresponding to a SCAN.Response service primitive. The function checks the data for completeness, forms and submits a ScanResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.6 DbvResourceReportResponse()

Purpose:

Send ResourceReportResponse-APDU.

Function Name and Parameters:

```
DbvResourceReportResponse (
DbvAssocId                    lAssocId,                  /* IN */
DbvResourceReportRsp        * psResourceReportRsp,       /* IN */
DbvError                    **ppsSIError                 /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psResourceReportRsp	Pointer to the data to be transferred. The definition of the data type *DbvResourceReportRsp* corresponds to the definition of the RESOURCE-REPORT.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psResourceReportRsp* the data corresponding to a RESOURCE-REPORT.Response service primitive. The function checks the data for completeness, forms and submits a ResourceReportResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.7 DbvSortResponse()

Purpose:

Send SortResponse-APDU.

Function Name and Parameters:

```
DbvSortResponse (
DbvAssocId        lAssocId,       /* IN */
DbvSortRsp      * psSortRsp,      /* IN */
DbvError       **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psSortRsp	Pointer to the data to be transferred. The definition of the data type *DbvSortRsp* corresponds to the definition of the SORT.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psSortRsp* the data corresponding to a SORT.Response service primitive. The function checks the data for completeness, forms and submits a SortResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.8 DbvSegmentationRequest()

Purpose:

Send SegmentationRequest-APDU.

Function Name and Parameters:

```
DbvSegmentationRequest (
DbvAssocId                  lAssocId,                /* IN */
DbvSegmentationReq        * psSegmentationReq,       /* IN */
DbvError                 **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psSegmentationReq	Pointer to the data to be transferred. The definition of the data type *DbvSegmentationReq* corresponds to the definition of the SEGMANTATION.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psSegmentationReq* the data corresponding to a SEGMENTATION.Request service primitive. The function checks the data for completeness, forms and submits a SegmentationRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.2.4.9 DbvExtendedServicesResponse()

Purpose:

Send ExtendedServicesResponse-APDU.

Function Name and Parameters:

```
DbvExtendedServicesResponse (
DbvAssocId                     lAssocId,                  /* IN */
DbvExtendedServicesRsp        * psExtendedServicesRsp,    /* IN */
DbvError                      **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psExtendedServicesRsp	Pointer to the data to be transferred. The definition of the data type *DbvExtendedServicesRsp* corresponds to the definition of the EXTENDED-SERVICES.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psExtendedServicesRsp* the data corresponding to an EXTENDED-SERVICES.Response service primitive. The function checks the data for completeness, forms and submits an ExtendedServicesResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1.

3.2.4.10 DbvCloseRequest()

Purpose:

Send CloseRequest-APDU.

Function Name and Parameters:

```
DbvCloseRequest (
DbvAssocId          lAssocId,        /* IN */
DbvCloseReq        * psCloseReq,     /* IN */
DbvError          **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

LAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
PsCloseReq	Pointer to the data to be transferred. The definition of the data type *DbvCloseReq* corresponds to the definition of the CLOSE.Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psCloseReq* the data corresponding to a CLOSE.Request service primitive. The function checks the data for completeness, forms and submits a CloseRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.2.4.11 DbvCloseResponse()

Purpose:

Send CloseResponse-APDU.

Function Name and Parameters:

```
DbvCloseResponse (
DbvAssocId        lAssocId,      /* IN */
DbvCloseReq       * psCloseRsp,  /* IN */
DbvError          **ppsSIError   /* OUT_F there is only one APDU defined for CLOSE */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvReceiveAssociateRequest().
psCloseRsp	Pointer to the data to be transferred. The definition of the data type *DbvCloseReq* corresponds to the definition of the CLOSE.Response service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psCloseRsp* the data corresponding to a CLOSE.Response service primitive. The function checks the data for completeness, forms and submits a CloseResponse-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.2.4.1

3.2.4.12 DbvAccessControlRequest()

Purpose:

Send Access Control Request APDU.

Function Name and Parameters:

```
DbvAccessControlRequest (
DbvAssocId                 lAssocId,                /* IN */
DbvAccessControlReq      * psAccessControlReq,      /* IN */
DbvError                 **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psAccessControlReq	Pointer to the data to be transferred. The definition of the data type *DbvAccessControlReq* corresponds to the definition of the ACCESS CONTROL Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psAccessControlReq* the data corresponding to an ACCESS CONTROL Request service primitive. The function checks the data for completeness, forms and submits an AccessControlRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.2.4.13 DbvResourceControlRequest()

Purpose:

Send Resource Control Request APDU.

Function Name and Parameters:

```
DbvResourceControlRequest (
DbvAssocId                  lAssocId,                 /* IN */
DbvResourceControlReq      * psResourceControlReq,    /* IN */
DbvError                  **ppsSIError                /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association, as received by a call of function DbvAssociateRequest().
psResourceControlRsp	Pointer to the data to be transferred. The definition of the data type *DbvResourceControlReq* corresponds to the definition of the RESOURCE CONTROL Request service primitive.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *psResourceControlRsp* the data corresponding to an RESOURCE CONTROL Request service primitive. The function checks the data for completeness, forms and submits an ResourceControlRequest-APDU.

If the function returns SI_OK, the data had been submitted successfully.

If the function returns SI_NOTOK, the data had not been transferred. In this case the *DbvError* structure indicates the reason for the transfer failure.

Usage: Refer to 3.1.3.1.

3.2.4.14 DbvReceiveDataTarget()

Purpose:

Receive any SR-/Z39.50 data.

Used by:

Target-System.

Function Name and Parameters:

```
DbvReceiveDataTarget (
DbvAssocId          lAssocId,          /* IN */
DbvSecs             lSeconds,          /* IN */
DbvTargetData       ** ppsTargetData,  /* OUT_F */
DbvError            **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association.
lSeconds	A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately, i.e. if there are currently no data available, the function does not wait for any incoming data. If the function returns SI_OK, then the pointer associated with parameter *ppsTargetData* either was not updated, in which case no data had been available, or refers to a filled-in *DbvTargetData* structure which provides the received data. A value ***CALL_BLOCKING*** causes the function to wait for incoming data (or any event indicating a failure). If the function returns SI_OK, the pointer associated with *ppsTargetData* refers to the received data. Any other value greater than zero causes the function to wait the given number of seconds for the incoming of any data (or any other event indicating a failure). If the function returns SI_OK, then the pointer associated with parameter *ppsTargetData* either was not updated, in which case no data were available or refers to a filled-in *DbvTargetData* structure which provides the received data.

ppsTargetData	The address of a pointer of type **DbvTargetData*. When calling the function, the pointer should refer to (DbvTargetData*)NULL. The pointer is updated only if the function returns SI_OK. In this case the pointer refers to a filled-in *DbvTargetData* structure.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function is used to receive SR-/Z39.50 requests, submitted by the "Origin System".

If the function returns SI_NOTOK, an error occured and component *eAssociationState* of the error structure indicates whether or not the association still exists.

The meaning of a return value SI_OK depends on the value of *lSeconds* (see description above).

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvTargetData            *psTargetData = NULL;
DbvError                 *psError = NULL;


if (DbvReceiveDataTarget(lAssocId, CALL_BLOCKING, &psTargetData, &psError) ==
SI_OK)
{
   /* process data */
   SIFreeDbvTargetData(&psTargetData);
             :
             :
}
else
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
}
```

See also:

Functions described in chapters 3.1.3.1 - 3.1.3.12.

3.3 Utility Functions

3.3.1 SIGetEventLocation()

Purpose:

Notify the API-user if there are any data to be received.

Function Name and Parameters:

```
SIGetEventLocation (
DbvAssocId          **pplAssocId,              /* IN */
DbvSecs               lSeconds,                /* IN */
DbvFds              **ppulFileDescriptor ,     /* IN */
DbvTargetData       ** ppsEventLocation,       /* OUT_F */
DbvError            **ppsSIError               /* OUT_F */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

pplAssocId — A list of association identifiers.

lSeconds — A value ***CALL_ASYNCHRONOUS*** causes the function to return immediately if there are currently no data to be received. If the function returns SI_OK, then the value which parameter *peEventLocation* refers either was not updated, in which case no data are to be received, or indicates the location where data are available.
A value ***CALL_BLOCKING*** causes the function to wait for incoming data (or any event indicating a failure). If the function returns SI_OK, the value which parameter *peEventLocation* refers to indicates the location where data are available.
Any other value greater than zero causes the function to wait the given number of seconds for the incoming of any data (or any other event indicating a failure). If the function returns SI_OK, then the value which parameter *peEventLocation* either was not updated, in which case no data are available or indicates the location where data are available.

ppulFileDescriptor	A list of file descriptors. The list shall be terminated by an entry (DbvFds*) NULL. Each file descriptor of this list is used on top of the API for any communication purpose.
ppsEventLocation	The address of a pointer of type **DbvEventLocation*.When calling the function, the pointer should refer to (DbvEventLocation*)NULL. The function has to watch the file descriptors provided in the list which parameter *ppulFileDescriptor* refers to (the API-user file descriptors) and additionally all those file descriptors used below the API which are related to any of the association identifiers parameter *pplAssocId* refers to (the API-provider file descriptors) for any incoming data. If the function perceives any incoming data, the pointer is updated. The structure the pointer refers to provides a list of file descriptors and/or a list of association identifiers.In the first case there are any data to be received on top of the API, in the second case there are data to be received below the API. In the first case the API-user may call an appropriate "private" function to read the data. In the second case the user has to call function DbvReceiveDataOrigin() with an appropriate association identifier (DbvReceiveDataTarget() respectively) to receive the data available below the API.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *pplAssocId* a list of association identifiers and in parameter *ppulFileDescriptor* one or more file descriptors. Each association identifier uniquely identifies a currently active association. Each file descriptor of the list is used for any communication purpose on top of the API.

The function has to watch the file descriptors provided in the list which *ppulFileDescriptor* refers to (the API-user file descriptors) and additionally all file descriptors used below the API which are related to any of the association identifiers provided in parameter *pplAssocId* (the API-provider file descriptors) for any incoming data.

If the function perceives any incoming data, a structure *DbvEventLocation* provides a list of file descriptors and/or a list of association identifiers.In the first case there are any data to be received on top of the API, in the second case there are data to be received below the API. In the first case the API-user may call an appropriate "private" function to read the data. In the second case the user has to call function DbvReceiveDataOrigin() with an appropriate association identifier (DbvReceiveDataTarget() respectively) to receive the data available below the API.

If the function returns SI_NOTOK, an error occured. In this case the *DbvError* structure indicates the error reason.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               **pplAssocId = NULL;
DbvFds                     ulFdsHost;
DbvFds                   **ppulFds = NULL;
DbvOriginData              *psOriginData = NULL;
DbvEventLocation           *psEventLocation = NULL;
DbvError                   *psError = NULL;
/* suppose  there is one active association, ulFdsHost is used for the
communication with
   the host and there is already  a value assigned to ulFdsHost. */

pplAssocId = (DbvAssocId**)malloc(2*sizeof(DbvAssocId*));
pplAssocId[0] = /* identifier of active association */;
pplAssocId[1] = (DbvAssocId*)NULL;

ppulFds = (Fds**)malloc(2*sizeof(Fds*));
ppulFds[0] = &ulFdsHost;
ppulFds[1] = (Fds*)NULL;

          :
          :
if (SIGetEventLocation(pplAssocId, CALL_BLOCKING, ppulFds, &psEventLocation,
&psError) == SI_OK)
{
   if (psEventLocation->ppulUserEvent != NULL)
   {
      /*
          read data from host
      */
   }
   if (psEventLocation ->pplProviderEvent != NULL)
   {
        foreach association identifier in psEventLocation->pplProviderEvent:
        {
           call function   DbvReceiveDataOrigin()  (function
DbvReceiveDataTarget() respectively)
         }
    }
 }
else
{
    PrivateErrorHandling(psError);
    FreeErrorCode(&psError);
}
```

3.3.2 SISetACSEFile()

Purpose:

Provide the name of a file which shall be used as a substitute for the default "ACSE-configuration-file" (refer to chapter 4.4).

Function Name and Parameters:

SISetACSEFile (

Char	*pcACSEFile,	/* IN */
DbvError	**ppsSIError	/* OUT_F */

)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

PcACSEFile	The name of the file providing the values for an AAssociate.Request service primitive.
PpsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The caller of the function provides in parameter *pcACSEFile* the name of a file which shall be used by function DbvAssociateRequest() as a substitute for the default "ACSE configuration file". The syntax of this file shall be according to the default "ACSE-configuration-file".

The function checks the existence of the file but does not check the content of the file. An invalid file content will be detected by function DbvAssociateRequest(), when trying to fill in the AAssociate.Request service primitive.

If the file exists and no other error occured, the function returns SI_OK.

If the function returns SI_NOTOK, an error occured. In this case the *DbvError* structure indicates the error reason.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget                *psTargetId = (DbvTarget*)NULL;
DbvAssociateState         eAssocState = Associate_Null;
DbvAssociateResult       *puAssociateResult = NULL;
DbvError                 *psError = NULL;
char                     *pcTargetSpecificACSE = "ACSETarget1"


if (SISetACSEFile(pcTargetSpecificACSE, &psError) == SI_OK)
{
   if (DbvAssociateRequest(&lAssocId, CALL_BLOCKING, psTargetId, &eAssocState,
        &puAssociateResult, &psError) == SI_OK)
   {
      :
      :
  }
}
else
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
           :
           :
}
```

See Also:

DbvAssociateRequest() using the ACSE-services.

3.3.3 SIWhichStack()

Purpose:

Get the type of stack in use for the current association/connection.

Function Name and Parameters:

```
SIWhichStack (
DbvAssocId              lAssocId,            /* IN */
DbvProtocolStack       * peProtocolStack,     /* OUT_C */
DbvError              **ppsSIError           /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association.
peProtocolStack	A pointer to a value of type *DbvProtocolStack*. The value should be ***Unknown_Stack*** when calling the function. The value is updated only if the function returns SI_OK. In this case the value indicates the protocol stack currently in use, i.e. ***OSI_Stack*** or ***TCP_Stack***.
PpsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

If the function returns SI_OK, the value which parameter *peProtocolStack* refers to indicates the protocol stack in use for the association/connection identified by *lAssocId*.

If the function returns SI_NOTOK, an error occured. In this case the *DbvError* structure indicates the error reason.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
DbvTarget                *psTargetId = (DbvTarget*)NULL;
DbvAssociateState         eAssocState = Associate_Null;
DbvAssociateResult       *puAssociateResult = NULL;
DbvProtocolStack          eStack = Unknown_Stack;
DbvError                 *psError = NULL;


if (DbvAssociateRequest(&lAssocId, CALL_BLOCKING, psTargetId, &eAssocState,
                     &puAssociateResult, &psError) == SI_OK)
{
   if (eAssocState == Associate_Accepted)
   {
      if (puAssociateResult->psProtocolAndSyntaxes->peApplicationProtocol ==
Unknown_Protocol)
      {
         if (SIWhichStack(lAssocId, &eStack, &psError) == SI_NOTOK)
         {
            /*     PrivateErrorHandling(psError);
                           :
         {
         else if (eStack == OSI_Stack)
         {
             /*  bug in function DbvAssociateRequest()   */
         }
         else   /* TCP_Stack */
         {
            /* the protocol to be used must be known */
         }
      }
      else
      {
         /*
             check wether or not the values in DbvProtocolAndSyntaxes
             are acceptable
         */
                 :
      }
      FreeAssociateResult(Associate_Accepted, &puAssociateResult);
   }
   else if (eAssocState == Associate_Rejected)
   {
      if (SIWhichStack(lAssocId, &eStack, &psError) == SI_NOTOK)
      {
         /*     PrivateErrorHandling(psError);
                        :
      {
      else if (eStack == TCP_Stack)
      {
         /*  bug in function DbvAssociateRequest()   */
      }
```

```
      else   /* OSI_Stack */
      {

         /*
             Inspect puAssociateResult->psAssociateReject.
             notify user */
         */
      }
      FreeAssociateResult(Associate_Rejected, &puAssociateResult);
   }
   else
   {
       /* bug in function DbvAssociateRequest() */
   }
}
else
{
   PrivateErrorHandling(psError);
                 :
                 :
}

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

See Also:

DbvAssociateRequest().

3.3.4 SISetTraceParameter()

Purpose:

Specify the destination for trace information and the trace depth for all API-functions.

Function Name and Parameters:

```
SISetTraceParameter (
DbvAssocId          lAssocId,        /* IN */
FILE              * psTraceFp,       /* IN */
TraceLevel          eTraceLevel,     /* IN */
DbvError          **ppsSIError       /* OUT_F */
)
```

Return Value:

SI_OK:	if no error occured.
SI_NOTOK:	on error.

Parameter Description:

lAssocId	The unique identifier of the association.
psTraceFp	The file pointer associated with the file, the trace information is to be written.
eTraceLevel	The trace level.
ppsSIError	The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL. The pointer is updated only if the function returns SI_NOTOK. In this case the pointer refers to a filled-in *DbvError* structure providing information about the failure occured.

Function Description:

The function receives a file pointer in parameter *psTraceFp*. The file pointer is associated with a file into which trace information provided by the API- and API-user functions is to be written. Parameter *eTraceLevel* indicates the required trace level.

If the function returns SI_NOTOK, an error occured. In this case the *DbvError* structure indicates the error reason.

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId = NO_ASSOCIATION_ID_ASSIGNED;
FILE                     *psTraceFp = NULL;
DbvTraceLevel             eTraceLevel = Trace_Level_3;
DbvError                 *psError = NULL;

/* open trace file */
if ((psTraceFp = fopen("MyTraceFile", "w")) == NULL)
{
    /* error handling */
}
else if (SISetTraceParameter(lAssocId, psTraceFp, eTraceLevel, &psError) ==
SI_NOTOK)
{
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
          :
          :
}
else
{
           :
           :
}
```

3.4 Utility Functions to Release DBV Datastructure Memory

The following utility functions have been provided from version 2.0 of the API software. These utility functions will release memory that has been allocated to the different Dbv data structures.

Note that memory of any sub-structures is also released.

There are four groups of functions for releaseing memory:

- A single function to release memory in the structure that has been allocated by the API function DbvReceiveDataOrigin() containing data of an APDU received from the Target;
- A group of functions to release memory in the structures that have been allocated by the application containing data to be sent to the Target;
- A single function to release memory in the structure that has been allocated by the API function DbvReceiveDataTarget() containing data of an APDU received from the Origin;
- A group of functions to release memory in the structures that have been allocated by the application containing data to be sent to the Origin.

3.4.1 Releasing Memory in a PDU Received at the Origin

Chapter 3.4.1.1 describes the function to release memory in the structure that has been allocated by the API function DbvReceiveDataOrigin()

3.4.1.1 SIFreeDbvOriginData()

Purpose:

To release memory allocated by the DbvReceiveDataOrigin() function.

Function Name and Parameters:

```
SIFreeDbvOriginData (
DbvOriginData                    **ppsOriginData                /* IN */
)
```

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsOriginData The origin data structure from which memory is to be released.

Function Description:

The caller of the function provides in parameter *ppsOriginData* the data structure that was returned from successfully calling function DbvReceiveDataOrigin(). The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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

if (DbvReceiveDataOrigin(lAssocId, CALL_BLOCKING, &psOriginData, &psError) ==
SI_OK)
{
   /* process the received data */
       :
       :
   /* Release the memory of the received data */
   if (SIFreeDbvOriginData(&psOriginData) != SI_OK)
   {
      HandleMemoryReleaseError();
   }
}
       :
       :
} else {
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
}
```

3.4.2 Releasing Memory for the Origin SR/Z39.50 Functions

Chapters 3.4.2.1 – 3.4.2.12 describe the functions to release memory used in the Origin SR/Z39.50 Functions.

3.4.2.1 SIFreeDbvInitializeReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvInitializeReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvInitializeReq (
DbvInitializeReq **ppsInitializeReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsInitializeReq	A pointer to the DbvInitializeReq data structure from which memory is to be released.

Function Description:

The caller of the function provides in parameter *ppsInitializeReq* the data structure that was created for an Initialize Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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

```
/*
    Allocate buffer for psInitializeReq and fill in structure.
*/


if (DbvInitializeRequest(lAssocId, psInitializeReq, &psError) == SI_OK)
{
   if (SIFreeInitializeReq(&psInitializeReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeInitializeReq(&psInitializeReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.2 SIFreeDbvSearchReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvSearchReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvSearchReq (
DbvSearchReq **ppsSearchReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsSearchReq A pointer to the DbvSearchReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsSearchReq* the data structure that was created for a Search Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psSearchReq and fill in structure.
*/
```

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

3.4.2.3 SIFreeDbvPresentReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvPresentReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvPresentReq (
DbvPresentReq **ppsPresentReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsPresentReq A pointer to the DbvPresentReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsPresentReq* the data structure that was created for a Present Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psPresentReq and fill in structure.
*/


if (DbvPresentRequest(lAssocId, psPresentReq, &psError) == SI_OK)
{
   if (SIFreePresentReq(&psPresentReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreePresentReq(&psPresentReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.4 SIFreeDbvDeleteResultSetReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvDeleteResultSetReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvDeleteResultSetReq (
DbvDeleteResultSetReq **ppsDeleteResultSetReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsDeleteResultSetReq A pointer to the DbvDeleteResultSetReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsDeleteResultSetReq* the data structure that was created for a Delete Result Set Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psDeleteResultSetReq and fill in structure.
*/


if (DbvDeleteResultSetRequest(lAssocId, psDeleteResultSetReq, &psError) ==
SI_OK)
{
```

```
   if (SIFreeDeleteResultSetReq(&psDeleteResultSetReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeDeleteResultSetReq(&psDeleteResultSetReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.5 SIFreeDbvScanReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvScanReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvScanReq (
DbvScanReq **ppsScanReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsScanReq A pointer to the DbvScanReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsScanReq* the data structure that was created for a Scan Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psScanReq and fill in structure.
*/


if (DbvScanRequest(lAssocId, psScanReq, &psError) == SI_OK)
{
   if (SIFreeScanReq(&psScanReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeScanReq(&psScanReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.6 SIFreeDbvResourceReportReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvResourceReportReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvResourceReportReq (
DbvResourceReportReq **ppsResourceReportReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsResourceReportReq A pointer to the DbvResourceReportReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsResourceReportReq* the data structure that was created for a Resource Report Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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

/*
    Allocate buffer for psResourceReportReq and fill in structure.
*/

if (DbvResourceReportRequest(lAssocId, psResourceReportReq, &psError) == SI_OK)
{
   if (SIFreeResourceReportReq(&psResourceReportReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeResourceReportReq(&psResourceReportReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.7 SIFreeDbvSortReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvSortReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvSortReq (
DbvSortReq **ppsSortReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsSortReq A pointer to the DbvSortReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsSortReq* the data structure that was created for a Sort Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psSortReq and fill in structure.
*/


if (DbvSortRequest(lAssocId, psSortReq, &psError) == SI_OK)
{
   if (SIFreeSortReq(&psSortReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeSortReq(&psSortReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.8 SIFreeDbvExtendedServicesReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvExtendedServicesReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvExtendedServicesReq (
DbvExtendedServicesReq **ppsExtendedServicesReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsExtendedServicesReq A pointer to the DbvExtendedServicesReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsExtendedServicesReq* the data structure that was created for a Extended Services Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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

/*
    Allocate buffer for psExtendedServicesReq and fill in structure.
*/

if (DbvExtendedServicesRequest(lAssocId, psExtendedServicesReq, &psError) ==
SI_OK)
```

```
{
   if (SIFreeExtendedServicesReq(&psExtendedServicesReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                  :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeExtendedServicesReq(&psExtendedServicesReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                  :
}
                  :
                  :
```

3.4.2.9 SIFreeDbvCloseReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvCloseReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvCloseReq (
DbvCloseReq **ppsCloseReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsCloseReq A pointer to the DbvCloseReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsCloseReq* the data structure that was created for a Close Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

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


/*
    Allocate buffer for psCloseReq and fill in structure.
*/


if (DbvCloseRequest(lAssocId, psCloseReq, &psError) == SI_OK)
{
   if (SIFreeCloseReq(&psCloseReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeCloseReq(&psCloseReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.10 SIFreeDbvAccessControlRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvAccessControlRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvAccessControlRsp (
DbvAccessControlRsp **ppsAccessControlRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsAccessControlRsp	A pointer to the DbvAccessControlRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsAccessControlRsp* the data structure that was created for an Access Control Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"


DbvAssocId                lAssocId;
DbvAccessControlRsp      *psAccessControlRsp = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psAccessControlRsp and fill in structure.
*/


if (DbvAccessControlResponse(lAssocId, psAccessControlRsp, &psError) == SI_OK)
{
   if (SIFreeAccessControlRsp(&psAccessControlRsp) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeAccessControlRsp(&psAccessControlRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.2.11 SIFreeDbvResourceControlRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvResourceControlRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvResourceControlRsp (
DbvResourceControlRsp **ppsResourceControlRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsResourceControlRsp A pointer to the DbvResourceControlRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsResourceControlRsp* the data structure that was created for an Resource Control Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"


DbvAssocId                lAssocId;
DbvResourceControlRsp    *psResourceControlRsp = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psResourceControlRsp and fill in structure.
*/


if (DbvResourceControlResponse(lAssocId, psResourceControlRsp, &psError) ==
SI_OK)
{
```

```
   if (SIFreeResourceControlRsp(&psResourceControlRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                  :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeResourceControlRsp(&psResourceControlRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                  :
}
                  :
                  :
```

3.4.2.12 SIFreeDbvTriggerResourceControlReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvTriggerResourceControlReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvTriggerResourceControlReq (
DbvTriggerResourceControlReq **ppsTriggerResourceControlReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsTriggerResourceControlReq	A pointer to the DbvTriggerResourceControlReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsTriggerResourceControlReq* the data structure that was created for a Trigger Resource Control Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                           lAssocId;
DbvTriggerResourceControlReq        *psTriggerResourceControlReq = NULL;
DbvError                            *psError = NULL;


/*
    Allocate buffer for psTriggerResourceControlReq and fill in structure.
*/
```

```
if (DbvTriggerResourceControlRequest(lAssocId, psTriggerResourceControlReq,
&psError) == SI_OK)
{
   if (SIFreeTriggerResourceControlReq(&psTriggerResourceControlReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeTriggerResourceControlReq(&psTriggerResourceControlReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.3 Releasing Memory in a PDU Received at the Target

Chapter 3.4.3.1 describes the function to release memory in the structure that has been allocated by the API function DbvReceiveDataTarget()

3.4.3.1 SIFreeDbvTargetData()

Purpose:

To release memory allocated by the DbvReceiveDataTarget() function.

Function Names and Parameters:

SIFreeDbvTargetData (
DbvTargetData **ppsTargetData /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsTargetData The target data structure from which memory is to be released.

Function Description:

The caller of the function provides in parameter *ppsTargetData* the data structure that was returned from successfully calling function DbvReceiveDataTarget(). The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

Usage:

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvTargetData            *psTargetData = NULL;
DbvError                 *psError = NULL;
```

```
if (DbvReceiveDataTarget(lAssocId, CALL_BLOCKING, &psTargetData, &psError) ==
SI_OK)
{
   /* process the received data */
       :
       :
   /* Release the memory of the received data */
   if (SIFreeDbvTargetData(&psTargetData) != SI_OK)
   {
      HandleMemoryReleaseError();
   }
}
       :
       :
} else {
   PrivateErrorHandling(psError);
   FreeErrorCode(&psError);
}
```

3.4.4 Releasing Memory for the Target SR/Z39.50 Functions

Chapters 3.4.4.1 – 3.4.4.10 describe the functions to release memory used in the Target SR/Z39.50 Functions.

3.4.4.1 SIFreeDbvInitializeRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvInitializeRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvInitializeRsp (
DbvInitializeRsp **ppsInitializeRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsInitializeRsp A pointer to the DbvInitializeRsp data structure from which memory is to be released.

Function Description:

The caller of the function provides in parameter *ppsInitializeRsp* the data structure that was created for an Initialize Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId              lAssocId;
DbvInitializeRsp       *psInitializeRsp = NULL;
DbvError               *psError = NULL;
```

```
/*
    Allocate buffer for psInitializeRsp and fill in structure.
*/


if (DbvInitializeResponse(lAssocId, psInitializeRsp, &psError) == SI_OK)
{
   if (SIFreeInitializeRsp(&psInitializeRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeInitializeRsp(&psInitializeRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.2 SIFreeDbvSearchRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvSearchRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvSearchRsp (
DbvSearchRsp **ppsSearchRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsSearchRsp A pointer to the DbvSearchRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsSearchRsp* the data structure that was created for a Search Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId              lAssocId;
DbvSearchRsp           *psSearchRsp = NULL;
DbvError               *psError = NULL;


/*
    Allocate buffer for psSearchRsp and fill in structure.
*/
```

```
if (DbvSearchResponse(lAssocId, psSearchRsp, &psError) == SI_OK)
{
   if (SIFreeSearchRsp(&psSearchRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                  :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeSearchRsp(&psSearchRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                  :
}
                  :
                  :
```

3.4.4.3 SIFreeDbvPresentRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvPresentRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvPresentRsp (
DbvPresentRsp **ppsPresentRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsPresentRsp A pointer to the DbvPresentRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsPresentRsp* the data structure that was created for a Present Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvPresentRsp            *psPresentRsp = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psPresentRsp and fill in structure.
*/


if (DbvPresentResponse(lAssocId, psPresentRsp, &psError) == SI_OK)
{
   if (SIFreePresentRsp(&psPresentRsp) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreePresentRsp(&psPresentRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.4 SIFreeDbvDeleteResultSetRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvDeleteResultSetRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvDeleteResultSetRsp (
DbvDeleteResultSetRsp **ppsDeleteResultSetRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsDeleteResultSetRsp A pointer to the DbvDeleteResultSetRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsDeleteResultSetRsp* the data structure that was created for a Delete Result Set Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                 lAssocId;
DbvDeleteResultSetRsp     *psDeleteResultSetRsp = NULL;
DbvError                  *psError = NULL;


/*
    Allocate buffer for psDeleteResultSetRsp and fill in structure.
*/


if (DbvDeleteResultSetResponse(lAssocId, psDeleteResultSetRsp, &psError) ==
SI_OK)
{
```

```
   if (SIFreeDeleteResultSetRsp(&psDeleteResultSetRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeDeleteResultSetRsp(&psDeleteResultSetRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.5 SIFreeDbvScanRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvScanRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvScanRsp (
DbvScanRsp **ppsScanRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsScanRsp A pointer to the DbvScanRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsScanRsp* the data structure that was created for a Scan Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"


DbvAssocId                lAssocId;
DbvScanRsp               *psScanRsp = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psScanRsp and fill in structure.
*/


if (DbvScanResponse(lAssocId, psScanRsp, &psError) == SI_OK)
{
   if (SIFreeScanRsp(&psScanRsp) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeScanRsp(&psScanRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.6 SIFreeDbvResourceReportRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvResourceReportRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvResourceReportRsp (
DbvResourceReportRsp **ppsResourceReportRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsResourceReportRsp A pointer to the DbvResourceReportRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsResourceReportRsp* the data structure that was created for a Resource Report Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"


DbvAssocId                lAssocId;
DbvResourceReportRsp     *psResourceReportRsp = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psResourceReportRsp and fill in structure.
*/


if (DbvResourceReportResponse(lAssocId, psResourceReportRsp, &psError) == SI_OK)
{
   if (SIFreeResourceReportRsp(&psResourceReportRsp) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeResourceReportRsp(&psResourceReportRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.7 SIFreeDbvSortRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvSortRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvSortRsp (
DbvSortRsp **ppsSortRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsSortRsp	A pointer to the DbvSortRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsSortRsp* the data structure that was created for a Sort Rspuest. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               lAssocId;
DbvSortRsp              *psSortRsp = NULL;
DbvError                *psError = NULL;


/*
    Allocate buffer for psSortRsp and fill in structure.
*/


if (DbvSortResponse(lAssocId, psSortRsp, &psError) == SI_OK)
{
   if (SIFreeSortRsp(&psSortRsp) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeSortRsp(&psSortRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.8 SIFreeDbvExtendedServicesRsp()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvExtendedServicesRsp. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvExtendedServicesRsp (
DbvExtendedServicesRsp **ppsExtendedServicesRsp /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsExtendedServicesRsp A pointer to the DbvExtendedServicesRsp data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsExtendedServicesRsp* the data structure that was created for a Extended Services Response. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvExtendedServicesRsp    *psExtendedServicesRsp = NULL;
DbvError                  *psError = NULL;


/*
    Allocate buffer for psExtendedServicesRsp and fill in structure.
*/


if (DbvExtendedServicesResponse(lAssocId, psExtendedServicesRsp, &psError) ==
SI_OK)
```

```
{
   if (SIFreeExtendedServicesRsp(&psExtendedServicesRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                  :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeExtendedServicesRsp(&psExtendedServicesRsp) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                  :
}
                  :
                  :
```

3.4.4.9 SIFreeDbvAccessControlReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvAccessControlReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvAccessControlReq (
DbvAccessControlReq **ppsAccessControlReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsAccessControlReq	A pointer to the DbvAccessControlReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsAccessControlReq* the data structure that was created for an Access Control Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId                lAssocId;
DbvAccessControlReq      *psAccessControlReq = NULL;
DbvError                 *psError = NULL;


/*
    Allocate buffer for psAccessControlReq and fill in structure.
*/


if (DbvAccessControlRequest(lAssocId, psAccessControlReq, &psError) == SI_OK)
{
   if (SIFreeAccessControlReq(&psAccessControlReq) != SI_OK) {
```

```
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeAccessControlReq(&psAccessControlReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

3.4.4.10 SIFreeDbvResourceControlReq()

Purpose:

To release memory allocated in and beneath a datastructure of type DbvResourceControlReq. This function will also release all sub-structures that are beneath the elements in this structure.

Function Name and Parameters:

SIFreeDbvResourceControlReq (
DbvResourceControlReq **ppsResourceControlReq /* IN */
)

Return Value:

SI_OK: if no error occured.

SI_NOTOK: on error.

Parameter Description:

ppsResourceControlReq A pointer to the DbvResourceControlReq data structure from which memory is to be freed.

Function Description:

The caller of the function provides in parameter *ppsResourceControlReq* the data structure that was created for an Resource Control Request. The memory allocated to any substructures that are carried will also be released (such as Externals).

The function will return SI_OK if successful else will return SI_NOTOK

```
#include "sidefs.h"
#include "sitypes.h"
#include "sitypes2.h"
#include "dbvproto.h"
#include "iord_ext.h"
#include "itemordr.h"

DbvAssocId               lAssocId;
DbvResourceControlReq   *psResourceControlReq = NULL;
DbvError                *psError = NULL;


/*
    Allocate buffer for psResourceControlReq and fill in structure.
*/


if (DbvResourceControlRequest(lAssocId, psResourceControlReq, &psError) ==
SI_OK)
{
```

```
   if (SIFreeResourceControlReq(&psResourceControlReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
                 :
}
else
{
   PrivateErrorHandling(psError);
   if (SIFreeResourceControlReq(&psResourceControlReq) != SI_OK) {
   {
       HandleMemoryReleaseError();
   }
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                 :
}
                 :
                 :
```

Annex A: Reference List

The following documents are appropriate for the DBV OSI II Project Partners:

[1] D-022 SIS (Software Internal Specification)

[2] D-023 Test Tool Manual

[3] D-024 Installation and Operations

[4] ANSI Z39.50-1995 Search and Retrieval Standard

Annex B: The Default "ACSE-configuration-file"

The default "ACSE-configuration-file" provides the values of those ACSE parameters, which shall be present in the AAssociate.Request and AAssociateResponse service primitive upon association establishment.

The file provides values for the following ACSE parameters:

Application Context Name	Permitted values: "Basic_SR_Application_Context" and/or "Basic_Z3950_Application_Context". The application on top of the API may support both protocols, SR and Z39.50. In this case both of the above values shall be present. Default value: "Basic_SR_Application_Context".
Presentation Context Definition List	A list of abstract record- and format syntaxes (in "dot"-notation) supported by the application on top of the API. Default value Origin: Empty list.

Example File:

```
Application_Context_Name,   "Basic_Z3950_Application_Context",
                            "Basic_SR_Application_Context";

Presentation_Context_Definition_List,    "1.2.840.10003.5.1",   /* UNIMARC */
                                   "1.2.840.10003.5.101",       /* SUTRS */
                                   "1.2.840.10003.7.1";         /* Resource-Report-1 */
```

For more details refer to [1].

The ACSE-configuration-file is accessed by the functions DbvAssociateRequest(), DbvReceiveAssociateRequest() and DbvAssociateResponse. In case the file is missing or its content is disrupted, the behaviour of these functions is as described below.

DbvAssociateRequest():

1. ACSE_Configuration_File is missing or no ACN value provided or no PCDL provided

function returns	SI_OK
function uses	default value Basic_SR_Application_Context and omits PCDL in the AssociateRequest APDU
function writes	warning to log-file

statemachine transission to state DBV_OS_ASSOC_INPROGRESS

2. ACSE_Configuration_File exists, invalid ACN value(s)

   function returns SI_NOTOK, "SI_Interface_Error", "DBV_ACSE_INVALID_ACN_VALUE"

   statemachine remains in state DBV_OS_CLOSED

3. ACSE_Configuration_File exists, incorrect format

   function returns SI_NOTOK, "SI_Interface_Error", "DBV_ACSE_FORMATERROR"

   statemachine remains in state DBV_OS_CLOSED

DbvReceiveAssociateRequest():

1. File is missing or file is disrupted (incorrect format) or no PCDL value provided

   function returns SI_NOTOK, "SI_Interface_Error", "DBV_ACSE_MISSING_FILE" or "DBV_ACSE_FORMATERROR" or "DBV_ACSE_MISSING_PCDL_VALUE"

   if the Origin provided a PCDL in the AssociateRequest, the function rejects any abstract syntaxes in the list (i.e. indicates "No_Common_Syntax"). In the following, function DbvAssociateResponse() implicitly changes the service-user provided value "eAssociateRsp" to "Rejected_Trans"!

   statemachine remains in state DBV_TS_CLOSED, accepts AssociateResponse(**-**) or Abort.

   if the Origin didn't provide a PCDL, the function also indicates "No_Common_Syntax". Function DbvAssociateResponse() however leaves the service-user provided value "eAssociateRsp" unchanged!

   statemachine transition to state DBV_TS_ASSOC_INPROGRESS, i.e. accepts AssociateResponse(+/-) or Abort.

2. File exists, no ACN value provided

   function returns SI_OK

   function uses default value Basic_SR_Application_Context

   function writes warning to log-file

   statemachine transition to state DBV_TS_ASSOC_INPROGRESS, i.e. accepts AssociateResponse(+/-) or Abort.

3. File exists, invalidACN value(s) provided

   function returns SI_NOTOK, "SI_Interface_Error", "DBV_ACSE_INVALID_ACN_VALUE

   statemachine transition to state DBV_TS_ASSOC_INPROGRESS, i.e. accepts AssociateResponse(+/-) or Abort.

DbvAssociateResponse():

1. File is missing or file is disrupted (incorrect format) or no PCDL value provided

   if the Origin provided a PCDL in the AssociateRequest and the service-user tries to accept the association:

   function returns SI_NOTOK, "SI_Interface_Error", "DBV_ACSE_MISSING_FILE" or "DBV_ACSE_FORMATERROR" or "DBV_ACSE_MISSING_PCDL_VALUE"

   statemachine remains in state DBV_TS_CLOSED

   the function implicitly changes the service-user provided value "eAssociateRsp" to "Rejected_Trans" and indicates in the Error structure (parameter "eAssociationState") that the association does not exist "Association_Closed"

2. Any other case of missing or invalid ACSE related values

   function returns SI_OK

   error was already indicated by function DbvReceiveAssociateRequest().