**Projekt: DBV-OSI II**

# - API-Definition -

Dok.-Nr.: D-008

Version: 5.0

Verantwortlicher Bearbeiter

B.Hergeth, Fa. Danet

Printed : 18.10.1995

# Inhaltsverzeichnis

**1. Einleitung ..... 7**

**2. Allgemeine Beschreibung des API ..... 9**
2.1 Allgemeine Beschreibung der API-Funktionen ..... 9
2.2 Namenskonventionen ..... 10
2.3 Bereitstellung der API-Funktionen ..... 11

**3. API-Functions ..... 12**
3.1 Functions Used by the Origin System ..... 12
3.1.1 Association Handling Functions Using the ACSE-Services ..... 12
3.1.1.1 DbvAssociateRequest() ..... 13
3.1.1.2 DbvReleaseRequest() ..... 17
3.1.1.3 DbvAbortRequest() ..... 20
3.1.1.4 DbvConfirmAssociateRequest() ..... 22
3.1.1.5 DbvConfirmReleaseRequest() ..... 26
3.1.2 Association Handling Functions Using TCP-Services ..... 28
3.1.2.1 DbvAssociateRequest() ..... 28
3.1.2.2 DbvReleaseRequest() ..... 32
3.1.2.3 DbvAbortRequest() ..... 34
3.1.2.4 DbvConfirmAssociateRequest() ..... 36
3.1.2.5 DbvConfirmReleaseRequest() ..... 36
3.1.3 SR-/Z39.50 Functions ..... 37
3.1.3.1 DbvInitializeRequest() ..... 37
3.1.3.2 DbvSearchRequest() ..... 39
3.1.3.3 DbvPresentRequest() ..... 40
3.1.3.4 DbvDeleteResultSetRequest() ..... 41
3.1.3.5 DbvScanRequest() ..... 42
3.1.3.6 DbvResourceReportRequest() ..... 43
3.1.3.7 DbvSortRequest() ..... 44
3.1.3.8 DbvExtendedServicesRequest() ..... 45
3.1.3.9 DbvCloseRequest() ..... 46
3.1.3.10 DbvCloseResponse() ..... 47
3.1.3.11 DbvReceiveDataOrigin() ..... 48
3.2 Functions Used by the Target System ..... 50
3.2.0 DbvTargetInitialize() ..... 50
3.2.1 Association Handling Functions Using the ACSE-Services ..... 51
3.2.1.1 DbvReceiveAssociateRequest() ..... 51
3.2.1.2 DbvAssociateResponse() ..... 55
3.2.1.3 DbvReleaseResponse() ..... 57
3.2.1.4 DbvAbortRequest() ..... 59
3.2.2 Association Handling Functions Using TCP-Services ..... 60
3.2.2.1 DbvReceiveAssociateRequest() ..... 60
3.2.2.2 DbvAssociateResponse() ..... 62
3.2.2.3 DbvReleaseResponse() ..... 64
3.2.2.4 DbvAbortRequest() ..... 66
3.2.3 SR-/Z39.50 Functions ..... 67
3.2.3.1 DbvInitializeResponse() ..... 67
3.2.3.2 DbvSearchResponse() ..... 69

3.2.3.3 DbvPresentResponse() ..... 70
3.2.3.4 DbvDeleteResultSetResponse() ..... 71
3.2.3.5 DbvScanResponse() ..... 72
3.2.3.6 DbvResourceReportResponse() ..... 73
3.2.3.7 DbvSortResponse() ..... 74
3.2.3.8 DbvSegmentationRequest() ..... 75
3.1.3.9 DbvExtendedServicesResponse() ..... 76
3.2.3.10 DbvCloseRequest() ..... 77
3.2.3.11 DbvCloseResponse() ..... 78
3.2.3.12 DbvReceiveDataTarget() ..... 79
3.3 Utility Functions ..... 81
3.3.1 SIGetEventLocation() ..... 81
3.3.2 SISetACSEFile() ..... 84
3.3.3 SIWhichStack() ..... 86
3.3.4 SISetTraceParameter() ..... 89

**Annex A: DBV-OSI II Datatypes ..... 91**

**Annex B: The Default "ACSE-configuration-file" ..... 139**

**Referenzliste ..... 143**

# Verzeichnis der Abbildungen

Abbildung 1 "OSI-Umgebung" ..... 7

Abbildung 2 "TCP/IP-Umgebung" ..... 7

Abbildung 3 Struktur Funktionsbibliotheken (Beispiel Origin) ..... 11

# 1. Einleitung

Das vorliegende Dokument beschreibt das DBV-OSI II API (Application Programmers Interface).

Das API realisiert die Schnittstelle zwischen dem "SR-/Z39.50 service-user" und dem "SR-/Z39.50 service-provider".

Die "SR/Z39.50 service-provider" SW ist in zwei *Umgebungen* verfügbar:

- der von den Protokollen SR und Z39.50 geforderten OSI-Umgebung, d.h. der Verbindungsauf- und -abbau (und -abbruch) wird über die Dienste des "Association Control Service Elements" (ACSE), der eigentliche Datentransfer über die Dienste der OSI-Presentation-Ebene realisiert (vgl. Abbildung 1)

- in einer nicht protokollkonformen TCP/IP-Umgebung, in der der "SR-/Z39.50 service-provider" direkt auf dem "Transmission Control Protocol" (TCP) aufsetzt (vgl. Abbildung 2).

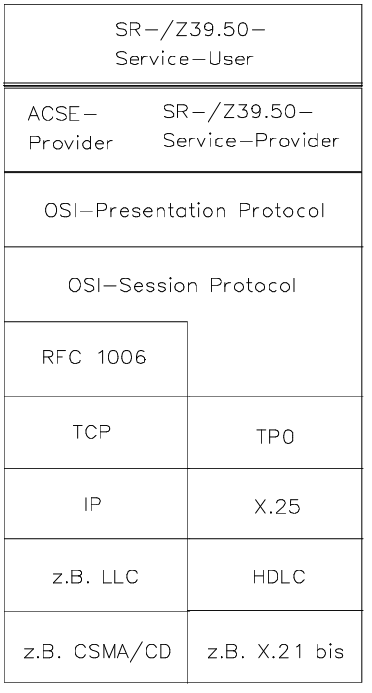


Abbildung 1: "OSI–Umgebung"

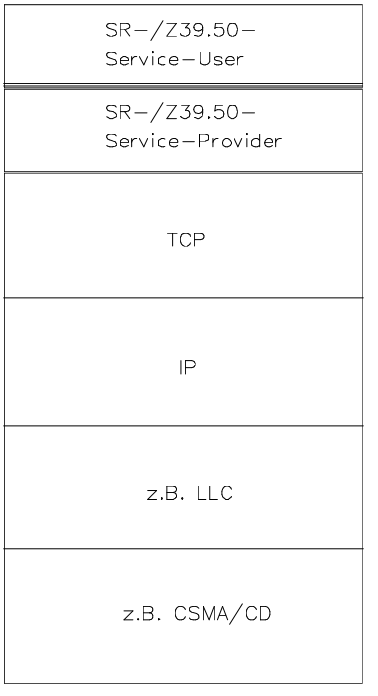


Abbildung 2: "TCP/IP–Umgebung"

Die bereitgestellten API-Funktionen lassen sich unterteilen in

- Funktionen zur Realisierung des Verbindungsauf- und -abbaus (und Verbindungsabbruchs)

- Funktionen zur Realisierung des eigentlichen SR-/Z39.50 Datentransfers.

Aus Sicht des API-Benutzers, ist das Verhalten der Funktionen zur Realisierung des eigentlichen Datentransfers unabhängig von der Umgebung, d.h. das "Coding" für den Aufruf der Funktionen und die Behandlung der Rückgabewerte ist für beide Umgebungen identisch.

Für die Funktionen zur Realisierung des Verbindungsauf- und -abbaus ergeben sich (leichte) Unterschiede bzgl. der Behandlung der von der jeweiligen Funktion gelieferten Rückgabewerte. In Kapitel 3 wird an den entsprechenden Stellen auf die Unterschiede hingewiesen. Falls die Anwendung der Funktionen unabhängig von der Umgebung sein soll, d.h. identisches "Coding" für beide Umgebungen, müssen die Unterschiede bzgl. der Behandlung der Rückgabewerte im Anwenderprogramm behandelt werden. In Abschnitt 3.3.3 (Anwendung der Funktion SIWhichStack()) findet sich ein Beispiel für den umgebungsunabhängigen Aufruf der Funktion DbvAssociateRequest().

In Kapitel 2 wird das API zunächst allgemein beschrieben.

In Kapitel 3 erfolgt die Beschreibung der API-Funktionen. Da dieser Teil des Dokuments ggf. als Teil eines Benutzerhandbuchs übernommen werden kann, erfolgen sämtliche Beschreibungen innerhalb des Kapitels in englischer Sprache. Das gleiche gilt für die Ausführungen in Anhang A und Anhang B.

Anhang A beschreibt die DBV-OSI II Datentypen.

Anhang B beschreibt den Aufbau des "ACSE-configuration-files".

# 2. Allgemeine Beschreibung des API

In diesem Kapitel erfolgt eine allgemeine Beschreibung des DBV-OSI II API.

Die Abschnitte 2.1 und 2.2 beschreiben die generellen Eigenschaften der in Kapitel 3 definierten Funktionen und erläutern die bei der Definition der Funktionsparameter angewandten Namenskonventionen. Abschnitt 2.3 beschreibt in welcher Form die API-Funktionen dem Benutzer zur Verfügung stehen.

## 2.1 Allgemeine Beschreibung der API-Funktionen

Das API stellt

- Funktionen zur Realisierung des Verbindungsaufbaus-, -abbaus und -abbruchs
- Funktionen zur Realisierung der SR-/Z39.50 Daten-Übertragung und
- einige nicht-kommunikationsorientierte Hilfsfunktionen (utility functions)

zur Verfügung.

Jede Funktion liefert als "Return Value" entweder den Wert **SI_OK** oder den Wert **SI_NOTOK**.

Der Wert SI_OK zeigt an, daß die Funktion ohne Fehler ausgeführt wurde.

Der Wert SI_NOTOK zeigt an, daß bei der Ausführung der Funktion ein Fehler auftrat. Der aufgetretene Fehler wird in einer Datenstruktur angezeigt, die wie folgt definiert ist:

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

Der Parameter *eSIErrorClass* gibt die Fehlerklasse an. Mögliche Werte sind:

| | |
|---|---|
| SI_Communication_Error | z.B. fehlerhafte PDU etc. |
| SI_Interface_Error | z.B. Funktion mit ungültigen Parametern aufgerufen etc. |
| SI_OperatingSystem_Error | z.B. Datei kann nicht geöffnet werden, kein Speicherplatz mehr vorhanden etc. |
| SI_Abort | die Verbindung zum entfernten System existiert nicht mehr |

Im Falle von SI_Interface_Error, SI_Communication_Error und SI_OperatingSystem_Error, zeigt die Komponente *lIfOsCoError* der "union" den genauen Fehler an.

Im Falle SI_Abort wird der Fehler in der DbvAbort Datenstruktur angezeigt.

Der Parameter *eAssociationState* zeigt an, ob die Verbindung zum Partnersystem noch besteht oder nicht.

Bei der Funktionsbeschreibung sind die Funktionsparameter mit "IN" bzw. "OUT_C" oder "OUT_F" gekennzeichnet. Dabei bedeutet:

IN: Ein Wert dieses Parameters muß beim Aufruf an die Funktion übergeben werden.

OUT_C: Die Funktion weist diesem Parameter einen Wert zu. Der Speicherplatz für den Wert muß vom Aufrufer der Funktion bereitgestellt werden.

OUT_F: Die Funktion weist diesem Parameter einen Wert zu. Der Speicherplatz für den Wert wird von der Funktion bereitgestellt. Der Aufrufer der Funktion ist dafür verantwortlich, daß der Speicherplatz wieder freigegeben wird .

## 2.2 Namenskonventionen

Bei der Definition von Konstanten, Datenstrukturen, Elementen von "enumeration types" und Variablen wurden bei der Namensvergabe folgende Regeln angewandt:

| | |
|---|---|
| Konstanten: | Jedes Zeichen ist aus der Menge {A..Z, _} (z.B. SI_NOTOK). |
| Namen strukturierter Datentypen: | Jedes Zeichen ist aus der Menge {A..Z, a..z}, das erste Zeichen ist immer ein Großbuchstabe (z.B. SearchReq). |
| Elemente von "enumeration types": | Jedes Zeichen ist aus der Menge {A..Z, a..z, _, 1..9}, das erste Zeichen ist ein Großbuchstabe (z.B. Association_Accepted). |
| Variablen: | Jedes Zeichen ist aus der Menge {A..Z, a..z}, das erste Zeichen ist ein Kleinbuchstabe (z.B. psSearchReq). |

Bei der Benennung von Variablen wurde eine weitere Regel angewandt:

Jede Variable zeigt implizit ihren Typ an. Beispiele:

```
long          lAssocId;      /* l: long Variable */

long         *plAssocId;     /* pl: Pointer auf long Wert */

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

SearchReq    *psSearchReq;   /* ps: Pointer auf eine Datenstruktur */
```

```
DbvError        **ppsSIError;       /* pps: Pointer auf Pointer auf Datenstruktur */
```

## 2.3 Bereitstellung der API-Funktionen

Die API-Funktionen stehen in Form von Funktionsbibliotheken zur Verfügung. Für das "Binden" der Anwenderprogramme (das Erzeugen der Origin-/Target-Prozesse) existieren die erforderlichen "Makefiles".

Für jeden zu realisierenden Dienst existiert genau eine, "stack-unabhängige", API-Funktion. Diese Funktion stellt fest, über welchen Stack (siehe Abb. 1 und 2) eine Verbindung aufzubauen ist bzw. welcher Stack momentan "aktiv" ist und ruft dann die entsprechende OSI- bzw. TCP-spezifische Funktion auf.

Die "Struktur" der Funktionsbibliotheken ermöglicht es, die OSI- bzw. TCP-spezifische Funktionsbibliothek durch eine "Dummy"-Bibliothek zu ersetzen (vgl. Abb. 3).

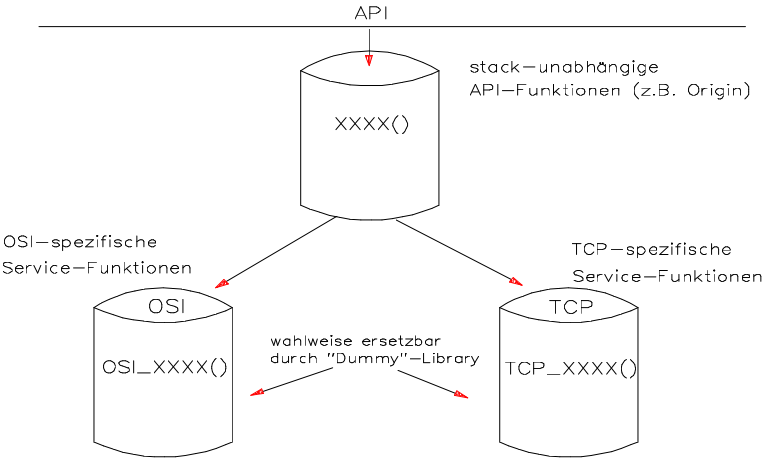


Abbildung 3: Struktur Funktionsbibliotheken (Beispiel Origin)

Hierdurch wird es möglich, stack-spezifische Prozesse zu erzeugen. Durch Ersetzen der OSI-spezifischen Bibliothek durch eine "Dummy"-Bibliothek ist es möglich einen TCP-spezifischen Prozess zu erzeugen. Durch Ersetzen der TCP-spezifischen Bibliothek durch eine "Dummy"-Bibliothek ist es möglich einen OSI-spezifischen Prozess zu erzeugen.

Weitere Information findet sich in [1].

# 3. API-Functions

This chapter describes the DBV-OSI II API.

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.

## 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.<br>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 "si_defs.h"
#include "si_types.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 "si_defs.h"
#include "si_types.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 occured.

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 "si_defs.h"
#include "si_types.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 "si_defs.h"
#include "si_types.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*.<br>The value should be ***Release_Null*** when calling the function.<br>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***.<br>***Release_Accepted*** indicates, that the association was released.<br>***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.<br>***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.<br>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_C */
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 "si_defs.h"
#include "si_types.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     *peAReleaseState,  /* 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*.<br>The value should be ***Release_Null*** when calling the function.<br>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.<br>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 "si_defs.h"
#include "si_types.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 "si_defs.h"
#include "si_types.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      **puAssociateResult,  /* OUT_C */
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 - 3.1.3.10 describe the functions available for sending SR-/Z39.50 data.

Chapter 3.1.3.11 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.<br>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 "si_defs.h"
#include "si_types.h"

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


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

    Allocate buffer for psInitializeReq and fill in structure.
*/


if (DbvInitializeRequest(lAssocId, psInitializeReq, &psError) == SI_OK)
{
   FreeInitializeReq(&psInitializeReq);
                :
                :
}
else
{
   PrivateErrorHandling(psError);
   FreeInitializeReq(&psInitializeReq);
   FreeErrorCode(&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.<br>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.<br>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.<br>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.<br>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.<br>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.<br>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 */
CloseReq            *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 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 "si_defs.h"
#include "si_types.h"

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


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

**See also:**

Functions described in chapters 3.2.3.1 - 3.2.3.10.

## 3.2 Functions Used by the Target System

Chapter 3.2.0 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 3.2.1 and 3.2.2 describe the association handling functions available to the API-user in the OSI-, and TCP-environment respectively.

Chapter 3.2.3 describes the functions avaiable for sending and receiving SR-/Z39.50 data in both environments.

### 3.2.0 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 |
| argv | argv parameter of the target main program |
| ppsSIError | The address of a pointer of type **DbvError*. When calling the function, the pointer should refer to (DbvError*)NULL.<br>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. |

For a more detailed description of this function refer to [1].

## 3.2.1 Association Handling Functions Using the ACSE-Services

The following functions are available to the API-user:

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

Chapters 3.2.1.1 - 3.2.1.4 describe these functions.

### 3.2.1.1 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.<br>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.<br>The pointer associated with parameter *ppcOriginAddress* refers to a character string indicating the address of the origin system that requested the association.<br>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.<br>If the function returns SI_OK, and there was no associate request the above pointers are not updated.<br>A value ***CALL_BLOCKING*** causes the function to wait for the incoming of an associate request.<br>If the function returns SI_OK, the pointer associated with parameter *ppcOriginAddress* and the pointer associated with parameter *ppsProtocolAndSyntaxes* are updated as described above.<br>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).<br>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.<br>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.<br>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*.<br>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.<br>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 "si_defs.h"
#include "si_types.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().

### 3.2.1.2 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 "si_defs.h"
#include "si_types.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().

### 3.2.1.3 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).<br>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.<br>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.<br>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 "si_defs.h"
#include "si_types.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 */
                  :
                  :
}
```

### 3.2.1.4 DbvAbortRequest()

Refer to 3.1.1.3.

## 3.2.2 Association Handling Functions Using TCP-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 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.<br>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<br>there was no transport connection established.<br>If the function returns SI_OK and there was a transport connection established, then<br>- 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.<br>- the pointer associated with parameter *ppsProtocolAndSyntaxes* is updated and refers to a filled in structure *DbvProtocolAndSyntaxes* (as described below*)*.<br>If the function returns SI_OK, and there was no connection established, the above pointers are not updated.<br>A value ***CALL_BLOCKING*** causes the function to wait for the incoming of a transport connect request.<br>If the function returns SI_OK, the pointers associated with parameters *ppcOriginAddress* and *ppsProtocolAndSyntaxes* are updated as described above.<br>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).<br>If the function returns SI_OK, the pointers associated with parameters *ppcOriginAddress* and *ppsProtocolAndSyntaxes* are updated as described above.<br>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.<br>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*.<br>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.<br>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 3.2.1.1.

### 3.2.2.2 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***.<br>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.<br>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 "si_defs.h"
#include "si_types.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 */
                :
                :
}

                :
                :
```

### 3.2.2.3 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 "si_defs.h"
#include "si_types.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 */
                 :
                 :
}
```

### 3.2.2.4 DbvAbortRequest()

Refer to 3.2.1.3.

## 3.2.3 SR-/Z39.50 Functions

Chapters 3.2.3.1 - 3.2.3.11 describe the functions available for sending SR-/Z39.50 data.

Chapter 3.1.3.12 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.

### 3.2.3.1 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.<br>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 "si_defs.h"
#include "si_types.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)
{
   FreeInitializeRsp(&psInitializeRsp);
            :
            :
}
else
{
   PrivateErrorHandling(psError);
   FreeInitializeRsp(&psInitializeRsp);
   FreeErrorCode(&psSIError); /* release buffer, assign NULL pointer */
                :
                :
}
              :
              :
```

### 3.2.3.2 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.<br>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 3.2.3.1

### 3.2.3.3 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.<br>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 3.2.3.1

### 3.2.3.4 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.<br>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 3.2.3.1

### 3.2.3.5 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.<br>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 3.2.3.1

### 3.2.3.6 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.<br>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 3.2.3.1

### 3.2.3.7 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.<br>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 3.2.3.1

### 3.2.3.8 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.<br>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.

### 3.1.3.9 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.<br>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 3.2.3.1.

### 3.2.3.10 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.<br>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.

### 3.2.3.11 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 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.<br>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

### 3.2.3.12 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.<br>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 "si_defs.h"
#include "si_types.h"

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


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

**See also:**

Functions described in chapters 3.1.3.1 - 3.1.3.9.

## 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 */
DbvEventLocation    **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.<br>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.<br>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.<br>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.<br>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.<br>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.<br>In the first case the API-user may call an appropriate "private" function to read the data.<br>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.<br>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 "si_defs.h"
#include "si_types.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.<br>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 "si_defs.h"
#include "si_types.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*.<br>The value should be ***Unknown_Stack*** when calling the function.<br>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.<br>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 "si_defs.h"
#include "si_types.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.<br>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 "si_defs.h"
#include "si_types.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
{
           :
           :
}
```

# Annex A: DBV-OSI II Datatypes

```
/* -----------  si_types.h ------------------------------------------*/


/* Responses for releasing an association */
typedef enum DbvReleaseRspReason {
      Normal = 0,
      Not_Finished = 1
} DbvReleaseRspReason;

typedef enum DbvReleaseRspResult {
      Affirmative = 0,
      Negative = 1
} DbvReleaseRspResult;

typedef struct DbvReleaseRsp {
      DbvReleaseRspReason      *peReleaseRspReason;
      DbvReleaseRspResult       eReleaseRspResult;
} DbvReleaseRsp;

/* Enumeration Types */
/* Events requiring a read operation:*/
typedef struct DbvEventLocation {
      DbvFds      **ppulUserEvent;
      DbvAssocId  **pplProviderEvent;
} DbvEventLocation;

/* Supported protocols: */
typedef enum DbvApplicationProtocol {
            Unknown_Protocol = 0,
            Unidentified_Protocol = 1,
            SR_Protocol = 2,
            Z3950_Protocol = 3
} DbvApplicationProtocol;
```

```
/* Supported protocol stacks: */
typedef enum DbvProtocolStack {
            Unknown_Stack = 0,
            OSI_Stack = 1,
            TCP_Stack = 2
} DbvProtocolStack;

/* Associate Response: */
typedef enum DbvAssociateRsp {
            Accepted = 0,
            Rejected_Perm = 1,
            Rejected_Tran = 2
} DbvAssociateRsp;

/* Abstract syntaxes: */
typedef enum DbvAbstractSyntax {
            No_Syntaxes_Provided = 0,
            No_Common_Syntax = 1,
            Common_Syntaxes = 2
} DbvAbstractSyntax;

/* Association states during association/connection establishment phase: */
typedef enum DbvAssociateState {
            Associate_Null  = 0,
            Associate_Accepted  = 1,
            Associate_Rejected  = 2,
            Associate_Pending  = 3
} DbvAssociateState;

/* Associate rejection: */
typedef enum DbvAssociateRejectSource {
            ACSE_User = 0,
            ACSE_Provider = 1
} DbvAssociateRejectSource;
```

```
typedef enum DbvAssociateRejectResult {
            Rejected_Permanent = 0,
            Rejected_Transient = 1
} DbvAssociateRejectResult;

typedef enum DbvAssociateRejectReason {
            No_Reason_Given = 0,
            No_Common_ACSE_Version = 1,
            Application_Context_Not_Supported = 2
} DbvAssociateRejectReason;

/* Association states during association/connection release phase: */
typedef enum DbvReleaseState {
            Release_Null  = 0,
            Release_Accepted  = 1,
            Release_Rejected  = 2,
            Release_Pending  = 3
} DbvReleaseState;

typedef enum DbvReleaseReq {
            ReleaseReq_Normal = 0,
            ReleaseReq_Urgent = 1,
            ReleaseReq_Undefined = 2
} DbvReleaseReq;

/* Association state upon return of an API function: */
typedef enum DbvAssociationState {
            Association_Open = 0,
            Association_Closed = 1,
            Association_Unknown_State = 2
} DbvAssociationState;

/* Error classes: */
typedef enum DbvErrorClass {
            SI_Communication_Error = 0,
            SI_Interface_Error = 1,
```

```
            SI_OperatingSystem_Error = 2,
            SI_Abort = 3
} DbvErrorClass;

/* Abort sources: */
typedef enum DbvAbortSource {
            SI_ACSE_User = 0,
            SI_ACSE_Provider = 1,
            SI_Presentation_User = 2,
            SI_Presentation_Provider = 3,
            SI_Transport_Provider = 4
} DbvAbortSource;

/* Permitted SR-/Z39.50 events at the Origin system */
typedef enum DbvOriginEvent {
            OR_InitializeConf = 0,
            OR_SearchConf = 1,
            OR_PresentConf = 2,
            OR_DeleteResultSetConf = 3,
            OR_ScanConf = 4,
            OR_ResourceReportConf = 5,
            OR_SegmentationInd = 6,
            OR_SortConf = 7,
            OR_ExtendedServicesConf = 8,
            OR_CloseConf = 9,
            OR_CloseInd = 10,
            OR_Invalid_APDU = 11
} DbvOriginEvent;

/* Permitted SR-/Z39.50 events at the Target system */
typedef enum DbvTargetEvent {
            TA_InitializeInd = 0,
            TA_SearchInd = 1,
            TA_PresentInd = 2,
            TA_DeleteResultSetInd = 3,
            TA_ScanInd = 4,
```

```
            TA_ResourceReportInd = 5,
            TA_SortInd = 6,
            TA_ExtendedServicesInd = 7,
            TA_CloseInd = 8,
            TA_CloseConf = 9,
            TA_ReleaseInd = 10,
            TA_Invalid_APDU = 11
} DbvTargetEvent;

/* Target identification */
typedef enum DbvTargetNameOrAddress {
            Target_Name = 0,
            Target_Address = 1
} DbvTargetNameOrAddress;

/* Address types */
typedef enum DbvOsiOrInternetAddress {
            Osi_Address = 0,
            Internet_Address = 1
} DbvOsiOrInternetAddress;

/* Trace level */
typedef enum DbvTraceLevel {
      Trace_Off = 0,
      Trace_Low = 1,
      Trace_High = 2
} DbvTraceLevel;


/* Structure/Union Types  */
/* Structures associated with Association Handling functions */
typedef struct DbvAssociateReject {
      DbvAssociateRejectSource      *peAssociateRejectSource;
      DbvAssociateRejectResult      *peAssociateRejectResult;
      DbvAssociateRejectReason      *peAssociateRejectReason;
} DbvAssociateReject;
```

```
typedef struct DbvProtocolAndSyntaxes {
      DbvApplicationProtocol          *peApplicationProtocol;
      DbvAbstractSyntax         *peAbstractSyntax;
      char                     **ppcCommonSyntaxes;
/* valid <--> peAbstractSyntax == Common_Syntaxes */
} DbvProtocolAndSyntaxes;

typedef union DbvAssociateResult {
      DbvProtocolAndSyntaxes        *psProtocolAndSyntaxes;
      DbvAssociateReject            *psAssociateReject;
} DbvAssociateResult;

/* OSI and TCP/IP adresses */
typedef struct DbvOsiAddress {
      char        *pcPsel;
      char        *pcSsel;
      char        *pcTsel;
      char        *pcNsap;
} DbvOsiAddress;

typedef struct DbvInternetAddress {
      char        *pcHostAddress;
      DbvPortNumber    ulPortNumber;
} DbvInternetAddress;

typedef struct DbvTargetInformation {
      DbvApplicationProtocol              *peTargetSupportedProtocol;
      DbvOsiOrInternetAddress               eOsiOrInternetAddress;
      union {
            DbvOsiAddress           *psOsiAddress;
            DbvInternetAddress      *psInternetAddress;
      }                               uOsiOrInternetAddress;
} DbvTargetInformation;
```

```
typedef struct DbvTarget {
      DbvTargetNameOrAddress              eTargetNameOrAddress;
      union {
            char              *pcTargetName;
            DbvTargetInformation    *psTargetInformation;
      }                               uNameOrAddress;
} DbvTarget;

/* Structures common to all functions */
typedef struct DbvAbort {
  DbvAbortSource          eAbortSource;
  DbvAbortReason          lAbortReason;
  char                         *pcDiagnostics;
} DbvAbort;

typedef struct DbvError {
  DbvErrorClass               eErrorClass;
  union {
    DbvErrorCode     lIfOsCoError;
    DbvAbort          *psAbort;
  }                     uErrorCode;
  DbvAssociationState         eAssociationState;
} DbvError;



/* ------------------------------------------------------------*/
/*               Types and Sub-Structures                      */
/* ------------------------------------------------------------*/
typedef unsigned long   Ulong;
typedef char*           DbvProtocolVersion;
typedef char*           DbvOptions;

typedef enum DbvBool {
             DBV_FALSE = 0,
             DBV_TRUE  = 1
} DbvBool;
```

```
typedef char*           DbvObjectIdentifier;

/* -------------   OctetString   ---------------------------- */
typedef struct DbvOctetString {
Ulong      ulLength;
char      *pcValue;
} DbvOctetString;

/* -------------   DbvVisibleString   ---------------------------- */
typedef char*          DbvVisibleString;

/* -------------   DbvGeneralizedTime   ---------------------------- */
typedef char*          DbvGeneralizedTime;

/* -------------   DbvInternationalString   ---------------------------- */
typedef struct DbvInternationalString {
Ulong      ulLength;
char      *pcValue;
} DbvInternationalString;

/* -------------   DbvGeneralString   ---------------------------- */
typedef struct DbvGeneralString {
Ulong      ulLength;
char      *pcValue;
} DbvGeneralString;

/* -------------  ASN1External  ---------------------- */
typedef enum DbvExternalValueType {
                    EVT_SingleASN1Type = 1,
                    EVT_OctetAligned   = 2,
                    EVT_Arbitrary      = 3
} DbvExternalValueType;

typedef enum DbvASN1Type {
             AT_UnstructuredText      = 1,
             AT_OpacRecord            = 2,
```

```
            AT_SummaryRecord         = 3,
            AT_ResourceReport1       = 4,
            AT_ResourceReport2       = 5,
            AT_EsTaskPackage         = 6,
            AT_UpdateExtendedService = 7,
            AT_UnknownType           = 8
} DbvASN1Type;

typedef struct DbvSingleASN1Type {
DbvASN1Type          eASN1Type;
union {
    struct DbvInternationalString   *psUnstructuredText;
    struct DbvOpacRecord            *psOpacRecord;
    struct DbvSummaryRecord         *psSummaryRecord;
    struct DbvResourceReport1       *psResourceReport1;
    struct DbvResourceReport2       *psResourceReport2;
    struct DbvEsTaskPackage         *psEsTaskPackage;
    struct DbvUpdateExtendedService *psUpdateExtendedService;
    struct DbvOctetString           *psUnknownType;
  }              uASN1;
} DbvSingleASN1Type;

typedef struct DbvASN1External {
DbvObjectIdentifier      pcObjectIdentifier;
DbvExternalValueType     eValueType;
union {
    DbvSingleASN1Type      *psSingleASN1Type;
    DbvOctetString         *psOctetAligned;
    DbvOctetString         *psArbitrary;
  }                  uExternalValue;
} DbvASN1External;

/* --------------  Other Information  ----------------------- */
typedef enum DbvInformationType {
                   IT_CharacterInfo    = 1,
                   IT_BinaryInfo       = 2,
```

```
                    IT_ExternalInfo     = 3,
                    IT_ObjectIdentifier = 4
} DbvInformationType;

typedef struct DbvInfoCategory {
DbvObjectIdentifier    pcCategoryTypeId;
Ulong                  ulCategoryValue;
} DbvInfoCategory;

typedef struct DbvOtherInformation {
DbvInfoCategory       *psInfoCategory;
DbvInformationType     eInformationType;
union {
    DbvInternationalString      *psCharacterInfo;
    DbvOctetString              *psBinaryInfo;
    DbvASN1External             *psExternallyDefinedInfo;
    DbvObjectIdentifier          pcObjectIdentifier;
  }                 uInformation;
} DbvOtherInformation;

/* --------------  Authentication ----------------------- */
typedef enum DbvAuthenticationType {
                  AT_Open       = 1,
                  AT_IdPassword = 2,
                  AT_Anonymous  = 3,
                  AT_Other      = 4
} DbvAuthenticationType;

typedef struct DbvIdPassword {
DbvInternationalString      *psGroupId;
DbvInternationalString      *psUserId;
DbvInternationalString      *psPassword;
} DbvIdPassword;

typedef struct DbvAuthentication{
DbvAuthenticationType   eAuthenticationType;
```

```
union {
    DbvVisibleString           psOpen;
    DbvIdPassword               *psIdPassword;
    DbvASN1External             *psOther;
  }                 uAuthentication;
} DbvAuthentication;

/* --------------  DbvElementSetNames  ---------------------- */
typedef enum DbvElementSetType {
                  EST_Generic          = 1,
                  EST_DatabaseSpecific = 2
} DbvElementSetType;

typedef struct DbvElementSetsDbSpecific {
DbvInternationalString       *psDatabaseName;
DbvInternationalString       *psElementSetName;
} DbvElementSetsDbSpecific;

typedef struct DbvElementSetNames {
DbvElementSetType      eElementSetType;
union {
    DbvInternationalString         *psGeneric;
    DbvElementSetsDbSpecific      **ppsElementSetsDbSpecific;
  }               uElementSet;
} DbvElementSetNames;

/* --------------  DbvQuery  ---------------------- */
typedef enum DbvQueryType {
             QT_Private         = 1,
             QT_Rpn             = 2,
             QT_Iso8777         = 3,
             QT_Z3958           = 4,
             QT_Erpn            = 5,
             QT_NaturalLanguage = 6
} DbvQueryType;
```

```
typedef  enum DbvRPNStructureType {
                  RST_RpnLopRopOperator = 1,
                  RST_RpnOperand        = 2
} DbvRPNStructureType;

typedef enum DbvOperandType {
                  OT_AttributePlusTerm       = 1,
                  OT_ResultSetId             = 2,
                  OT_ResultSetPlusAttributes = 3
} DbvOperandType;

typedef enum DbvTermType {
             TT_General          = 1,    /* the original, Version 2 type */
             TT_Numeric          = 2,
             TT_SimpleCharacter  = 3,
             TT_ObjectIdentifier = 4,
             TT_DateTime         = 5,
             TT_External         = 6,
             TT_IntegerAndUnit   = 7,
             TT_Null             = 8
} DbvTermType;

typedef enum DbvOperatorType {
                  OT_Or        = 1,
                  OT_And       = 2,
                  OT_AndNot    = 3,
                  OT_Proximity = 4
} DbvOperatorType;

typedef enum DbvProximityUnitType {
                     PUT_Known   = 1,
                     PUT_Private = 2
} DbvProximityUnitType;
```

```
typedef enum DbvProximityRelationType {
                        PRT_LessThan           = 1,
                        PRT_LessThanOrEqual    = 2,
                        PRT_Equal              = 3,
                        PRT_GreaterThanOrEqual = 4,
                        PRT_GreaterThan        = 5,
                        PRT_NotEqual           = 6,
                        PRT_Invalid            = -1
} DbvProximityRelationType;

typedef enum DbvKnownProximityUnit {
                      KPU_Character   = 1,
                      KPU_Word        = 2,
                      KPU_Sentence    = 3,
                      KPU_Paragraph   = 4,
                      KPU_Section     = 5,
                      KPU_Chapter     = 6,
                      KPU_Document    = 7,
                      KPU_Element     = 8,
                      KPU_SubElement  = 9,
                      KPU_ElementType = 10,
                      KPU_Byte        = 11,
                      KPU_Invalid     = -1
} DbvKnownProximityUnit;

typedef struct DbvProximityUnitCode {
DbvProximityUnitType    eProximityUnitType;
union {
    DbvKnownProximityUnit  eKnownProximityUnit;
    Ulong                  ulPrivate;
  }                  uProximityUnit;
} DbvProximityUnitCode;

typedef struct DbvProximityOperator {
DbvBool                   *pfExclusion;
Ulong                      ulDistance;
```

```
DbvBool                     fOrdered;
DbvProximityRelationType    eProximityRelationType;
DbvProximityUnitCode        sProximityUnitCode;
} DbvProximityOperator;

typedef struct DbvOperator {
DbvOperatorType       eOperatorType;
DbvProximityOperator *psProximityOperator; /* <--> eOperatorType == OT_Proximity
*/
} DbvOperator;

/* ---- DbvAttributeValue -------- */
typedef enum DbvAttributeValueType {
                AVT_Numeric  = 1,    /* the original one */
                AVT_Complex  = 2
} DbvAttributeValueType;

/* -------------   DbvIntUnit   -------------------- */
typedef enum DbvStringOrNumericType {
                 SNT_String  = 1,
                 SNT_Numeric = 2
} DbvStringOrNumericType;

typedef struct DbvStringOrNumeric {
DbvStringOrNumericType     eStringOrNumeric;
union {
    DbvInternationalString    *psString;
    Ulong                      ulNumeric;
  }                    uStringOrNumeric;
} DbvStringOrNumeric;

typedef struct DbvUnit {
DbvInternationalString *psUnitSystem;
DbvStringOrNumeric     *psUnitType;
DbvStringOrNumeric     *psUnit;
Ulong                  *pulScaleFactor;
} DbvUnit;
```

```
typedef struct DbvIntUnit {
Ulong          ulValue;
DbvUnit        sUnitUsed;
} DbvIntUnit;

typedef struct DbvComplexValue {
DbvStringOrNumeric    **ppsList;
Ulong                 **ppulSemanticAction;
} DbvComplexValue;

typedef struct DbvAttributeValue {
DbvAttributeValueType          eAttributeValueType;
union {
    Ulong                   ulNumeric;
    DbvComplexValue        *psComplexValue;
  }                             uAttributeValue;
} DbvAttributeValue;

typedef struct DbvAttributeElement {
DbvObjectIdentifier      pcAttributeSetId;
Ulong                    ulAttributeType;
DbvAttributeValue        sAttributeValue;
} DbvAttributeElement;

/* -------------   DbvTerm   ------------------ */
typedef struct DbvTerm {
DbvTermType        eTermType;
union {
    DbvOctetString              *psGeneral;
    Ulong                        ulInteger;
    DbvInternationalString      *psSimpleCharacter;
    DbvObjectIdentifier          pcObjectIdentifier;
    DbvGeneralizedTime           pcDateTime;
    DbvASN1External             *psExternal;
```

```
    DbvIntUnit                  *psIntegerAndUnit;
  }               uTerm;
} DbvTerm;

typedef struct DbvAttributesPlusTerm {
DbvAttributeElement    **ppsAttributeElement;
DbvTerm                  sTerm;
} DbvAttributesPlusTerm;

typedef struct DbvResultSetPlusAttributes {
DbvInternationalString       sResultSetId;
DbvAttributeElement        **ppsAttributeElement;
} DbvResultSetPlusAttributes;

typedef struct DbvOperand {
DbvOperandType   eOperandType;
union {
    DbvAttributesPlusTerm      *psAttributesPlusTerm;
    DbvInternationalString     *psResultSetId;
    DbvResultSetPlusAttributes *psResultSetPlusAttributes;
  }            uOperand;
} DbvOperand;

typedef struct DbvRPNLopRopOperator {
struct DbvRPNStructure  *psLeftOperand;
struct DbvRPNStructure  *psRightOperand;
DbvOperator              sOperator;
} DbvRPNLopRopOperator;

typedef struct DbvRPNStructure {
DbvRPNStructureType   eRpnStructureType;
union {
    DbvRPNLopRopOperator       *psRpnLopRopOperator;
    DbvOperand                 *psRpnOperand;
  }               uRPNStructure;
} DbvRPNStructure;
```

```
typedef struct DbvRPNQuery {
DbvObjectIdentifier         pcAttributeSetId;
DbvRPNStructure             sRpnStructure;
} DbvRPNQuery;

typedef struct DbvQuery {
DbvQueryType        eQueryType;
union {
    DbvOctetString          *psPrivateQuery;
    DbvRPNQuery             *psRpnQuery;
    DbvOctetString          *psIso8777Query;
    DbvOctetString          *psZ3958Query;
    DbvRPNQuery             *psErpnQuery;
    DbvOctetString          *psNaturalLanguageQuery;
  }             uQuery;
} DbvQuery;

/* --------------  DbvRecords  ----------------------- */
typedef enum DbvRecordType {
              RT_Response                   = 1,
              RT_NonSurrDiagnostics         = 2,
              RT_MultipleNonSurrDiagnostics = 3
} DbvRecordType;

typedef enum DbvDbRecordType {
                RT_Retrieval            = 1,
                RT_SurrogateDiagnostics = 2,
                RT_StartingFragment     = 3,
                RT_IntermediateFragment = 4,
                RT_FinalFragment        = 5
} DbvDbRecordType;

typedef enum DbvDiagnosticsType {
                  DT_DefaultFormat     = 1,
                  DT_ExternallyDefined = 2
} DbvDiagnosticsType;
```

```
typedef enum DbvFragmentSyntaxType {
                      FST_ExternallyTagged    = 1,
                      FST_NotExternallyTagged = 2
} DbvFragmentSyntaxType;

typedef enum DbvProtocolVersionInForce {
                      PVIF_Version2 = 1,
                      PVIF_Version3 = 2
} DbvProtocolVersionInForce;

typedef struct DbvAdditionalInfo {
DbvProtocolVersionInForce   eProtocolVersionInForce;
union {
    DbvVisibleString            pcV2;
    DbvInternationalString     *psV3;
  }                            uAdditionalInfo;
} DbvAdditionalInfo;

typedef struct DbvDefaultDiagFormat {
DbvObjectIdentifier        pcDiagnosticSetId;
Ulong                      ulCondition;
DbvAdditionalInfo          sAdditionalInfo;
} DbvDefaultDiagFormat;

typedef struct DbvDiagRec {
DbvDiagnosticsType       eDiagnosticsType;
union {
    DbvDefaultDiagFormat      *psDefault;
    DbvASN1External           *psExternallyDefined;
  }                  uDiagnostics;
} DbvDiagRec;
```

```
typedef struct DbvFragmentSyntax {
DbvFragmentSyntaxType      eFragmentSyntaxType;
union {
    DbvASN1External    *psExternallyTagged;
    DbvOctetString     *psNotExternallyTagged;
  }                     uFragmentSyntax;
} DbvFragmentSyntax;

typedef struct DbvNamePlusRecord {
DbvInternationalString  *psDatabaseName;
DbvDbRecordType          eDbRecordType;
union {
    DbvASN1External   *psRetrievalRecord;
    DbvDiagRec        *psSurrogateDiagnostic;
    DbvFragmentSyntax *psStartingFragment;
    DbvFragmentSyntax *psIntermediateFragment;
    DbvFragmentSyntax *psFinalFragment;
  }                   uDbRecord;
} DbvNamePlusRecord;

typedef struct DbvRecords {
DbvRecordType        eRecordType;
union {
    DbvNamePlusRecord    **ppsResponseRecord;
    DbvDefaultDiagFormat  *psNonSurrogateDiag;
    DbvDiagRec            **ppsMultipleNonSurrogateDiag;
  }               uRecord;
} DbvRecords;

/* --------------  DbvRange  ----------------------- */
typedef struct DbvRange {
Ulong        ulStartingPosition;
Ulong        ulNumberOfRecords;
} DbvRange;
```

```
/* --------------  RecordComposition  ---------------------- */
typedef enum DbvRecordCompositionType {
                          RCT_ElementSetNames = 1,
                          RCT_CompSpec        = 2
} DbvRecordCompositionType;

typedef enum DbvElementSpecType {
                   EST_Primitive = 1,
                   EST_External  = 2
} DbvElementSpecType;

typedef struct DbvSpecification {
DbvObjectIdentifier       pcSchema;
DbvElementSpecType        eElementSpecType;
union {
    DbvInternationalString  *psPrimitive;
    DbvASN1External          *psExternal;
  }                    uElementSpec;
} DbvSpecification;

typedef struct DbvDbSpecificComp {
DbvInternationalString    sDatabaseName;
DbvSpecification          sSpecification;
} DbvDbSpecificComp;

typedef struct DbvCompSpec {
DbvBool                    fSelectAlternativeSyntax;
DbvSpecification          *psGeneric;
DbvDbSpecificComp        **ppsDbSpecific;
DbvObjectIdentifier       *ppcRecordSyntax;
} DbvCompSpec;

typedef struct DbvRecordComposition {
DbvRecordCompositionType    eRecordCompositionType;
union {
```

```
    DbvElementSetNames  *psElementSetNames;
    DbvCompSpec         *psCompSpec;
  }                         uRecordComposition;
} DbvRecordComposition;

/* ----------  ListStatus  (--> DELETE-RESULT-SET response) --------- */
typedef enum DbvDeleteSetStatus {
                  DSS_Success                       = 0,
                  DSS_ResultSetDidNotExist          = 1,
                  DSS_PreviouslyDeletedByTarget     = 2,
                  DSS_SystemProblemAtTarget         = 3,
                  DSS_AccessNotAllowed              = 4,
                  DSS_ResourceControlAtOrigin       = 5,
                  DSS_ResourceControlAtTarget       = 6,
                  DSS_BulkDeleteNotSupported        = 7,
                  DSS_NotAllSetsDeletedOnBulkDlt    = 8,
                  DSS_NotAllRequestedSetsDeleted    = 9,
                  DSS_ResultSetInUse                = 10,
                  DSS_Invalid                       = -1
} DbvDeleteSetStatus;

typedef struct DbvListStatus {
DbvInternationalString       sResultSetId;
DbvDeleteSetStatus           eDeleteSetStatus;
} DbvListStatus;

/* --------------  DbvListEntries  (--> SCAN ) ---------------- */
typedef enum DbvEntryType {
            ET_TermInfo            = 1,
            ET_SurrogateDiagnostic = 2
} DbvEntryType;

typedef enum DbvOccurrenceType {
                  OT_Global     = 1,
                  OT_ByDatabase = 2
} DbvOccurrenceType;
```

```
typedef struct DbvOccurrenceByDb {
DbvInternationalString       sDatabaseName;
Ulong                        *pulNumber;
DbvOtherInformation         **ppsOtherInformation;
} DbvOccurrenceByDb;

typedef struct DbvOccurrence {
DbvOccurrenceType             eOccurrenceType;
union {
    Ulong                ulGlobal;
    DbvOccurrenceByDb  **ppsOccurrenceByDb;
  }                           uOccurence;
} DbvOccurrence;

typedef struct DbvOccurrenceByAttributes {
DbvAttributeElement      **ppsAttributeElement;
DbvOccurrence             *psOccurrence;
DbvOtherInformation      **ppsOtherInformation;
} DbvOccurrenceByAttributes;

typedef struct DbvTermInfo {
DbvTerm                      sTerm;
DbvInternationalString      *psDisplayTerm;
DbvAttributeElement        **ppsSuggestedAttribute;
DbvAttributesPlusTerm      **ppsAlternativeTerm;
Ulong                        *pulGlobalOccurrences;
DbvOccurrenceByAttributes  **ppsByAttributes;
DbvOtherInformation        **ppsOtherInformation;
} DbvTermInfo;

typedef struct DbvEntry {
DbvEntryType          eEntryType;
union {
    DbvTermInfo    *psTermInfo;
```

```
    DbvDiagRec     *psSurrogateDiagnostic;
  }               uEntry;
} DbvEntry;

typedef struct DbvListEntries {
DbvEntry    **ppsEntry;
DbvDiagRec  **ppsNonSurrogateDiagnostics;
} DbvListEntries;

/* --------------  SortSequence  ----------------------- */
typedef enum DbvSortElementType {
                     SET_Generic    = 1,
                     SET_DbSpecific = 2
} DbvSortElementType;

typedef enum DbvSortRelationType {
                    SRT_Ascending     = 1,
                    SRT_Descending    = 2,
                    SRT_FrequencyAsc  = 3,
                    SRT_FrequencyDesc = 4,
                    SRT_Invalid       = -1
} DbvSortRelationType;

typedef enum DbvSortCaseType {
                SCT_CaseSensitive   = 1,
                SCT_CaseInsensitive = 2,
                SCT_Invalid         = -1
} DbvSortCaseType;

typedef enum DbvSortKeyType {
                    SKT_SortField      = 1,
                    SKT_ElementSpec    = 2,
                    SKT_SortAttributes = 3
} DbvSortKeyType;
```

```
typedef enum DbvSortMVActionType {
                    SMT_Abort    = 1,
                    SMT_Null     = 2,
                    SMT_Constant = 3
} DbvSortMVActionType;

typedef struct DbvSortAttributes {
DbvObjectIdentifier     pcAttributeSetId;
DbvAttributeElement   **ppsAttributeElement;
} DbvSortAttributes;

typedef struct DbvSortKey {
DbvSortKeyType              eSortKeyType;
union {
    DbvInternationalString     *psSortField;
    DbvSpecification           *psElementSpec;
    DbvSortAttributes          *psSortAttributes;
  }                      uSortElement;
} DbvSortKey;

typedef struct DbvSortMVAction {
DbvSortMVActionType    eSortMVActionType;
DbvOctetString        *psOctetString;  /* <--> eSortMVActionType == SMT_Constant
*/
} DbvSortMVAction;

typedef struct DbvDbSpecificSortKey {
DbvInternationalString      sDatabaseName;
DbvSortKey                  sDbSort;
} DbvDbSpecificSortKey;

typedef struct DbvSortElement {
DbvSortElementType        eSortElementType;
union {
    DbvSortKey                *psGeneric;
```

```
    DbvDbSpecificSortKey     **ppsDatabaseSpecific;
  }                        uSortElement;
} DbvSortElement;

typedef struct DbvSortKeySpec {
DbvSortElement          sSortElement;
DbvSortRelationType     eSortRelationType;
DbvSortCaseType         eSortCaseType;
DbvSortMVAction        *psMVActionType;
} DbvSortKeySpec;

/* ---  OriginSuppliedCommonParameters  (--> EXTENDED-SERVICES) ----------- */
typedef enum DbvPermAllowableFunctions {
                      PAF_Delete            = 1,
                      PAF_ModifyContents    = 2,
                      PAF_ModifyPermissions = 3,
                      PAF_Present           = 4,
                      PAF_Invoke            = 5,
                      PAF_Invalid           = -1
} DbvPermAllowableFunctions;

typedef struct DbvPermissions {
DbvInternationalString          sUserId;
DbvPermAllowableFunctions     **ppeAllowableFunctions;
} DbvPermissions;
```

```
/* ------------------------------------------------------------------*/
/*                  PDU-Datastructures      (Top-Level structures)    */
/* ------------------------------------------------------------------*/

/* ------------------    INITIALIZE   ------------------------------- */
typedef struct DbvInitializeReq {
DbvOctetString         *psReferenceId;
DbvProtocolVersion      pcProtocolVersion;
DbvOptions              pcOptions;
Ulong                   ulPreferredMessageSize;
Ulong                   ulMaximumMessageSize;
DbvAuthentication      *psAuthentication;
DbvInternationalString *psImplementationId;
DbvInternationalString *psImplementationName;
DbvInternationalString *psImplementationVersion;
DbvASN1External        *psUserInformationField;
DbvOtherInformation   **ppsOtherInformation;
} DbvInitializeReq;

typedef struct DbvInitializeRsp {
DbvOctetString         *psReferenceId;
DbvProtocolVersion      pcProtocolVersion;
DbvOptions              pcOptions;
Ulong                   ulPreferredMessageSize;
Ulong                   ulMaximumMessageSize;
DbvBool                 fResult;
DbvInternationalString *psImplementationId;
DbvInternationalString *psImplementationName;
DbvInternationalString *psImplementationVersion;
DbvASN1External        *psUserInformationField;
DbvOtherInformation   **ppsOtherInformation;
} DbvInitializeRsp;
```

```
/* ------------------    SEARCH   ---------------------------------- */
typedef enum DbvResultSetStatus {
                    RSS_Subset  = 1,
                    RSS_Interim = 2,
                    RSS_None    = 3,
                    RSS_Invalid = -1
} DbvResultSetStatus;

typedef enum DbvPresentStatus {
                  PS_Success   = 0,
                  PS_Partial_1 = 1,
                  PS_Partial_2 = 2,
                  PS_Partial_3 = 3,
                  PS_Partial_4 = 4,
                  PS_Failure   = 5,
                  PS_Invalid   = -1
} DbvPresentStatus;

typedef struct DbvSearchReq {
DbvOctetString           *psReferenceId;
Ulong                     ulSmallSetUpperBound;
Ulong                     ulLargeSetLowerBound;
Ulong                     ulMediumSetPresentNumber;
DbvBool                   fReplaceIndicator;
DbvInternationalString    sResultSetName;
DbvInternationalString  **ppsDatabaseNames;
DbvElementSetNames       *psSmallSetElementNames;
DbvElementSetNames       *psMediumSetElementNames;
DbvObjectIdentifier       pcPreferredRecordSyntax;
DbvQuery                 *psQuery;
DbvOtherInformation     **ppsAdditionalSearchInfo;
DbvOtherInformation     **ppsOtherInformation;
} DbvSearchReq;
```

```
typedef struct DbvSearchRsp {
DbvOctetString         *psReferenceId;
Ulong                   ulResultCount;
Ulong                   ulNumberOfRecordsReturned;
Ulong                   ulNextResultSetPosition;
DbvBool                 fSearchStatus;
DbvResultSetStatus     *peResultSetStatus;
DbvPresentStatus       *pePresentStatus;
DbvRecords             *psRecords;
DbvOtherInformation   **ppsAdditionalSearchInfo;
DbvOtherInformation   **ppsOtherInformation;
} DbvSearchRsp;
```

```
/* ------------------    PRESENT   ---------------------------------- */
typedef struct DbvPresentReq{
DbvOctetString             *psReferenceId;
DbvInternationalString      sResultSetId;
Ulong                       ulResultSetStartPoint;
Ulong                       ulNumberOfRecordsRequested;
DbvRange                   **ppsAdditionalRanges;
DbvRecordComposition        *psRecordComposition;
DbvObjectIdentifier         pcPreferredRecordSyntax;
Ulong                       *pulMaxSegmentCount;
Ulong                       *pulMaxRecordSize;
Ulong                       *pulMaxSegmentSize;
DbvOtherInformation        **ppsOtherInformation;
} DbvPresentReq;

typedef struct DbvPresentRsp {
DbvOctetString         *psReferenceId;
Ulong                   ulNumberOfRecordsReturned;
Ulong                   ulNextResultSetPosition;
DbvPresentStatus        ePresentStatus;
DbvRecords             *psRecords;
DbvOtherInformation    **ppsOtherInformation;
} DbvPresentRsp;
```

```
/* ------------------    DELETE-RESULT-SET   ---------------------------- */
typedef enum DbvDeleteFunction {
                  DF_List     = 0,
                  DF_All      = 1,
                  DF_Invalid  = -1
} DbvDeleteFunction;

typedef struct DbvDeleteResultSetReq {
DbvOctetString               *psReferenceId;
DbvDeleteFunction             eDeleteFunction;
DbvInternationalString      **ppsResultSetId;
DbvOtherInformation         **ppsOtherInformation;
} DbvDeleteResultSetReq;

typedef struct DbvDeleteResultSetRsp {
DbvOctetString             *psReferenceId;
DbvDeleteSetStatus          eDeleteOperationStatus;
DbvListStatus              **ppsDeleteListStatuses;
Ulong                       *pulNumberNotDeleted;
DbvListStatus              **ppsBulkStatuses;
DbvInternationalString      *psDeleteMessage;
DbvOtherInformation        **ppsOtherInformation;
} DbvDeleteResultSetRsp;
```

```
/* ------------------    SEGMENT    ---------------------------- */
typedef struct DbvSegmentReq {
DbvOctetString         *psReferenceId;
Ulong                   ulNumberOfRecordsReturned;
DbvNamePlusRecord     **ppsSegmentRecords;
DbvOtherInformation   **ppsOtherInformation;
} DbvSegmentReq;


/* ------------------    RESOURCE-REPORT    ---------------------------- */
typedef enum DbvResourceReportStatus {
                        RRS_Success   = 0,
                        RRS_Partial   = 1,
                        RRS_Failure_1 = 2,
                        RRS_Failure_2 = 3,
                        RRS_Failure_3 = 4,
                        RRS_Failure_4 = 5,
                        RRS_Failure_5 = 6,
                        RRS_Failure_6 = 7,
                        RRS_Invalid   = -1
} DbvResourceReportStatus;

typedef struct DbvResourceReportReq {
DbvOctetString            *psReferenceId;
DbvOctetString            *psOperationId;
DbvObjectIdentifier        pcPrefResourceReportFormat;
DbvOtherInformation      **ppsOtherInformation;
} DbvResourceReportReq;

typedef struct DbvResourceReportRsp {
DbvOctetString             *psReferenceId;
DbvResourceReportStatus     eResourceReportStatus;
DbvASN1External            *psResourceReport;
DbvOtherInformation       **ppsOtherInformation;
} DbvResourceReportRsp;
```

```
/* ------------------    SCAN   ---------------------------- */
typedef enum DbvScanStatus {
              SCS_Success   = 0,
              SCS_Partial_1 = 1,
              SCS_Partial_2 = 2,
              SCS_Partial_3 = 3,
              SCS_Partial_4 = 4,
              SCS_Partial_5 = 5,
              SCS_Failure   = 6,
              SCS_Invalid   = -1
} DbvScanStatus;

typedef struct DbvScanReq {
DbvOctetString              *psReferenceId;
DbvInternationalString     **ppsDatabaseNames;
DbvObjectIdentifier          pcAttributeSetId;
DbvAttributesPlusTerm        sAttributesPlusTerm;
Ulong                       *pulStepSize;
Ulong                        ulNumberOfTermsRequested;
Ulong                       *pulPreferredPositionInResponse;
DbvOtherInformation        **ppsOtherInformation;
} DbvScanReq;

typedef struct DbvScanRsp {
DbvOctetString         *psReferenceId;
Ulong                  *pulStepSize;
DbvScanStatus           eScanStatus;
Ulong                   ulNumberOfEntriesReturned;
Ulong                  *pulPositionOfTerm;
DbvListEntries         *psEntries;
DbvObjectIdentifier     pcAttributeSetId;
DbvOtherInformation   **ppsOtherInformation;
} DbvScanRsp;
```

```
/* ------------------    SORT    ----------------------------- */
typedef enum DbvSortStatus {
                        SOS_Success   = 0,
                        SOS_Partial_1 = 1,
                        SOS_Failure   = 2,
                        SOS_Invalid   = -1
} DbvSortStatus;

typedef enum DbvSortResultSetStatus {
                        SRSS_Empty     = 0,
                        SRSS_Interim   = 1,
                        SRSS_Unchanged = 2,
                        SRSS_None      = 3,
                        SRSS_Invalid   = -1
} DbvSortResultSetStatus;

typedef struct DbvSortReq {
DbvOctetString           *psReferenceId;
DbvInternationalString  **ppsInputResultSetName;
DbvInternationalString    sSortedResultSetName;
DbvSortKeySpec          **ppsSortSequence;
DbvOtherInformation     **ppsOtherInformation;
} DbvSortReq;

typedef struct DbvSortRsp {
DbvOctetString             *psReferenceId;
DbvSortStatus               eSortStatus;
DbvSortResultSetStatus     *peResultSetStatus;
DbvDiagRec                **ppsDiagnostics;
DbvOtherInformation       **ppsOtherInformation;
} DbvSortRsp;
```

```
/* ------------------    CLOSE    ----------------------------- */
typedef enum DbvCloseReason {
               CR_Finished          = 0,
               CR_ShutDown          = 1,
               CR_SystemProblem     = 2,
               CR_CostLimit         = 3,
               CR_Resources         = 4,
               CR_SecurityViolation = 5,
               CR_ProtocolError     = 6,
               CR_LackOfActivity    = 7,
               CR_PeerAbort         = 8,
               CR_Unspecified       = 9,
               CR_Invalid           = -1
} DbvCloseReason;

typedef struct DbvCloseReq {
DbvOctetString                    *psReferenceId;
DbvCloseReason                     eCloseReason;
DbvInternationalString            *psDiagnosticInformation;
DbvObjectIdentifier                pcResourceReportFormat;
DbvASN1External                   *psResourceReport;
DbvOtherInformation              **ppsOtherInformation;
} DbvCloseReq;
```

```
/* ------------------    EXTENDED-SERVICES    ---------------------------- */
typedef enum DbvExtServicesFunctions {
                    ESF_Create = 1,
                    ESF_Delete = 2,
                    ESF_Modify = 3,
                    ESF_Invalid      = -1
} DbvExtServicesFunctions;

typedef enum DbvExtServicesWaitAction {
                    ESWA_Wait              = 1,
                    ESWA_WaitIfPossible    = 2,
                    ESWA_DontWait          = 3,
                    ESWA_DontReturnPackage = 4,
                    ESWA_Invalid           = -1
} DbvExtServicesWaitAction;

typedef enum DbvExtServicesOperationStatus {
                    ESOS_Done    = 1,
                    ESOS_Queued  = 2,
                    ESOS_Failure = 3,
                    ESOS_Invalid = -1
} DbvExtServicesOperationStatus;

typedef struct DbvExtendedServicesReq {
DbvOctetString                      *psReferenceId;
DbvExtServicesFunctions              eFunction;
DbvObjectIdentifier                  pcPackageType;
DbvInternationalString              *psPackageName;
DbvInternationalString              *psUserId;
DbvIntUnit                          *psRetentionTime;
DbvPermissions                     **ppsPermissions;
DbvInternationalString              *psDescription;
DbvASN1External                     *psOriginSpecificParameter;
DbvExtServicesWaitAction             eWaitAction;
DbvInternationalString              *psElements;
```

```
DbvOtherInformation                **ppsOtherInformation;
} DbvExtendedServicesReq;

typedef struct DbvExtendedServicesRsp {
DbvOctetString                     *psReferenceId;
DbvExtServicesOperationStatus       eOperationStatus;
DbvDiagRec                         **ppsDiagnostics;
DbvASN1External                    *psTaskPackage;
DbvOtherInformation                **ppsOtherInformation;
} DbvExtendedServicesRsp;
```

```
/* ------------------------------------------------------------*/
/*   Object Identifier, Record Syntaxes, Resource Report      */
/* ------------------------------------------------------------*/

/* -------------- Object Identifier (Record Syntaxes) -----------*/

#define ISO_UNIMARC             "1.0.10163.5.1"
#define ANSI_UNIMARC            "1.2.840.10003.5.1"
#define ISO_INTERMARC           "1.0.10163.5.2"
#define ANSI_INTERMARC          "1.2.840.10003.5.2"
#define ISO_CCF                 "1.0.10163.5.3"
#define ANSI_CCF                "1.2.840.10003.5.3"
#define ISO_USMARC              "1.0.10163.5.10"
#define ANSI_USMARC             "1.2.840.10003.5.10"
#define ISO_UKMARC              "1.0.10163.5.11"
#define ANSI_UKMARC             "1.2.840.10003.5.11"
#define ISO_NORMARC             "1.0.10163.5.12"
#define ANSI_NORMARC            "1.2.840.10003.5.12"
#define ISO_LIBRISMARC          "1.0.10163.5.13"
#define ANSI_LIBRISMARC         "1.2.840.10003.5.13"
#define ISO_DANMARC             "1.0.10163.5.14"
#define ANSI_DANMARC            "1.2.840.10003.5.14"
#define ISO_FINMARC             "1.0.10163.5.15"
#define ANSI_FINMARC            "1.2.840.10003.5.15"
#define ISO_MAB                 "1.0.10163.5.16"
#define ANSI_MAB                "1.2.840.10003.5.16"
#define ISO_CANMARC             "1.0.10163.5.17"
#define ANSI_CANMARC            "1.2.840.10003.5.17"
#define ISO_SBN                 "1.0.10163.5.18"
#define ANSI_SBN                "1.2.840.10003.5.18"
#define ISO_PICAMARC            "1.0.10163.5.19"
#define ANSI_PICAMARC           "1.2.840.10003.5.19"
#define ISO_AUSMARC             "1.0.10163.5.20"
#define ANSI_AUSMARC            "1.2.840.10003.5.20"
```

```
#define ISO_UNSTRUCTURED_TEXT_RECORD   "1.0.10163.5.101"
#define ANSI_UNSTRUCTURED_TEXT_RECORD  "1.2.840.10003.5.101"
#define ISO_OPAC_RECORD                "1.0.10163.5.102"
#define ANSI_OPAC_RECORD               "1.2.840.10003.5.102"
#define ISO_SUMMARY_RECORD             "1.0.10163.5.103"
#define ANSI_SUMMARY_RECORD            "1.2.840.10003.5.103"
#define ISO_ES_TASK_PACKAGE_RECORD     "1.0.10163.5.106"
#define ANSI_ES_TASK_PACKAGE_RECORD    "1.2.840.10003.5.106"

/* -------------- Object Identifier (Attribute-Set) -----------*/
#define ISO_ATTRIBUTE_SET_BIB_1      "1.0.10163.3.1"
#define ANSI_ATTRIBUTE_SET_BIB_1     "1.2.840.10003.3.1"

/* -------------- Object Identifier (Diagnostic-Set) -----------*/
#define ISO_DIAGNOSTIC_SET_BIB_1      "1.0.10163.4.1"
#define ANSI_DIAGNOSTIC_SET_BIB_1     "1.2.840.10003.4.1"

/* -------------- Object Identifier (Resource-Report) -----------*/
#define ISO_RESOURCE_REPORT_1        "1.0.10163.7.1"
#define ANSI_RESOURCE_REPORT_1       "1.2.840.10003.7.1"
#define ISO_RESOURCE_REPORT_2        "1.0.10163.7.2"
#define ANSI_RESOURCE_REPORT_2       "1.2.840.10003.7.2"

/* -------------- Object Identifier (UPDATE Extended Service) -----------*/
#define ISO_UPDATE_EXTENDED_SERVICE        "1.0.10163.9.5"
#define ANSI_UPDATE_EXTENDED_SERVICE       "1.2.840.10003.9.5"

/* -------------- Record Syntax "UnstructuredText" ----------------*/
typedef DbvInternationalString   DbvUnstructuredText;

/* -------------- Record Syntax "OpacRecord" ----------------*/
typedef struct DbvCircRecord {
DbvBool                     fAvailableNow;
DbvInternationalString     *psAvailabilityDate;
DbvInternationalString     *psAvailableThru;
DbvInternationalString     *psRestrictions;
```

```
DbvInternationalString     *psItemId;
DbvBool                     fRenewable;
DbvBool                     fOnHold;
DbvInternationalString     *psEnumAndChron;
DbvInternationalString     *psMidspine;
DbvInternationalString     *psTemporaryLocation;
} DbvCircRecord;

typedef struct DbvVolume {
DbvInternationalString        *psEnumeration;
DbvInternationalString        *psChronology;
DbvInternationalString        *psEnumAndChron;
} DbvVolume;

typedef struct DbvHoldingsAndCircData {
DbvInternationalString        *psTypeOfRecord;
DbvInternationalString        *psEncodingLevel;
DbvInternationalString        *psFormat;
DbvInternationalString        *psReceiptAcqStatus;
DbvInternationalString        *psGeneralRetention;
DbvInternationalString        *psCompleteness;
DbvInternationalString        *psDateOfReport;
DbvInternationalString        *psNucCode;
DbvInternationalString        *psLocalLocation;
DbvInternationalString        *psShelvingLocation;
DbvInternationalString        *psCallNumber;
DbvInternationalString        *psShelvingData;
DbvInternationalString        *psCopyNumber;
DbvInternationalString        *psPublicNote;
DbvInternationalString        *psReproductionNote;
DbvInternationalString        *psTermsUseRepro;
DbvInternationalString        *psEnumAndChron;
DbvVolume                    **ppsVolume;
DbvCircRecord                **ppsCircRecord;
} DbvHoldingsAndCircData;
```

```
typedef enum DbvHoldingsRecType {
      HRT_MarcHoldingsRecord  = 1,
      HRT_HoldingsAndCirc     = 2
} DbvHoldingsRecType;

typedef struct DbvHoldingsRecord {
DbvHoldingsRecType      eHoldingsRecType;
union {
    DbvASN1External         sMarcHoldingsRecord;
    DbvHoldingsAndCircData  sHoldingsAndCirc;
  }                   uMarcOrHoldAndCirc;
} DbvHoldingsRecord;

typedef struct DbvOpacRecord {
DbvASN1External         *psBibliographicRecord;
DbvHoldingsRecord      **ppsHoldingsData;
} DbvOpacRecord;

/* -------------- Record Syntax "SummaryRecord" ----------------*/
typedef struct DbvFormatSpec {
DbvInternationalString          sType;
Ulong                          *pulSize;
Ulong                          *pulBestPosn;
} DbvFormatSpec;

typedef struct DbvSummaryRecord {
DbvInternationalString           sTitle;
DbvInternationalString          *psAuthor;
DbvInternationalString          *psCallNumber;
DbvInternationalString          *psRecordType;
DbvInternationalString          *psBibliographicLevel;
DbvFormatSpec                  **ppsFormat;
DbvInternationalString          *psPublicationPlace;
DbvInternationalString          *psPublicationDate;
DbvInternationalString          *psTargetSystemKey;
```

```
DbvInternationalString          *psSatisfyingElement;
Ulong                           *pulRank;
DbvInternationalString          *psDocumentId;
DbvInternationalString          *psAbstract;
DbvOtherInformation            **ppsOtherInformation;
} DbvSummaryRecord;

/* -------------- Record Syntax "Resource Report" ----------------*/
typedef enum DbvEstimateType {
                ET_CurrentSearchRecords    = 1,
                ET_FinalSearchRecords      = 2,
                ET_CurrentPresentRecords   = 3,
                ET_FinalPresentRecords     = 4,
                ET_CurrentOpTimeProcessing = 5,
                ET_FinalOpTimeProcessing   = 6,
                ET_CurrentAssocTime        = 7,
                ET_CurrentOperationCost    = 8,
                ET_FinalOperationCost      = 9,
                ET_CurrentAssocCost        = 10,
                ET_FinalOpTimeElapsed      = 11,
                ET_PercentComplete         = 12,
                ET_CurrentSearchAssocCost  = 13,
                ET_CurrentPresentAssocCost = 14,
                ET_CurrentConnectAssocCost = 15,
                ET_CurrentOtherAssocCost   = 16,
                ET_Invalid                 = -1
} DbvEstimateType;

typedef struct DbvEstimate1 {
DbvEstimateType          eEstimateType;
Ulong                    ulValue;
Ulong                   *pulCurrencyCode;
} DbvEstimate1;
```

```
typedef struct DbvResourceReport1 {
DbvEstimate1                **ppsEstimate;
DbvInternationalString       sMessage;
} DbvResourceReport1;

typedef struct DbvEstimate2 {
DbvStringOrNumeric       sType;
DbvIntUnit               sValue;
} DbvEstimate2;

typedef struct DbvResourceReport2 {
DbvEstimate2               **ppsEstimate;
DbvInternationalString       sMessage;
} DbvResourceReport2;

/* -------------- Record Syntax "Extended Services Task Package" ------------*/
typedef enum DbvExtServicesTaskStatus {
                ESTS_Pending  = 1,
                ESTS_Active   = 2,
                ESTS_Complete = 3,
                ESTS_Aborted  = 4,
                ESTS_Invalid  = -1
} DbvExtServicesTaskStatus;

typedef struct DbvEsTaskPackage {
DbvObjectIdentifier                 pcPackageType;
DbvInternationalString             *psPackageName;
DbvInternationalString             *psUserId;
DbvIntUnit                         *psRetentionTime;
DbvPermissions                     **ppsPermissions;
DbvInternationalString             *psDescription;
DbvOctetString                     *psTargetReference;
DbvGeneralizedTime                  pcCreationDateTime;
DbvExtServicesTaskStatus            eExtServicesTaskStatus;
DbvDiagRec                         **ppsPackageDiagnostics;
```

```
DbvASN1External                         sTaskSpecificParameters;
} DbvEsTaskPackage;

/* -------------  UPDATE  EXTENDED SERVICE ---------------- */
typedef enum DbvUpdtAction {
                          UA_RecordInsert  = 1,
                          UA_RecordReplace = 2,
                          UA_RecordDelete  = 3,
                          UA_ElementUpdate = 4,
                          UA_Invalid       = -1
} DbvUpdtAction;

typedef enum DbvUpdtTaskStatus {
                  UTS_Success  = 1,
                  UTS_Partial  = 2,
                  UTS_Failure  = 3,
                  UTS_Invalid  = -1
} DbvUpdtTaskStatus;

typedef enum DbvUpdtRecordStatus {
                  URST_Success   = 1,
                  URST_Queued    = 2,
                  URST_InProcess = 3,
                  URST_Failure   = 4,
                  URST_Invalid   = -1
} DbvUpdtRecordStatus;

typedef enum DbvUpdtRecordOrDiagType {
                  URDT_Record     = 1,
                  URDT_Diagnostic = 2
} DbvUpdtRecordOrDiagType;


typedef enum DbvUpdtSupplementalIdType {
                USIT_Time            = 1,
                USIT_Version         = 2,
```

```
                USIT_PreviousVersion = 3
} DbvUpdtSupplementalIdType;

typedef enum DbvUpdtRecordIdType {
                URIT_Number = 1,
                URIT_String = 2,
                URIT_Opaque = 3
} DbvUpdtRecordIdType;

typedef struct DbvUpdtRecordOrDiag {
DbvUpdtRecordOrDiagType      eUpdtRecordOrDiag;
union {
    DbvASN1External      *psRecord;
    DbvDiagRec           *psDiagnostic;
  }                          uUpdtRecordOrDiag;
} DbvUpdtRecordOrDiag;

typedef struct DbvCorrelationInfo {
DbvInternationalString  *psNote;
Ulong                   *pulId;
} DbvCorrelationInfo;

typedef struct DbvTaskPackageRecord {
DbvUpdtRecordOrDiag       *psUpdtRecordOrDiag;
DbvCorrelationInfo        *psCorrelationInfo;
DbvUpdtRecordStatus        eUpdtRecordStatus;
} DbvTaskPackageRecord;

typedef struct DbvUpdateTargetPart {
DbvUpdtTaskStatus           eTaskStatus;
DbvDiagRec                **ppsGlobalDiagnostics;
DbvTaskPackageRecord      **ppsTaskPackageRecord;
} DbvUpdateTargetPart;

typedef struct DbvUpdtSupplementalId {
DbvUpdtSupplementalIdType     eSupplementalIdType;
```

```
union {
    DbvGeneralizedTime      pcTime;
    DbvInternationalString *psVersion;
    DbvASN1External        *psPreviousVersion;
  }                           uUpdtSupplementalId;
} DbvUpdtSupplementalId;

typedef struct DbvUpdtRecordId {
DbvUpdtRecordIdType   eRecordIdType;
union {
    Ulong                   ulNumber;
    DbvInternationalString *psString;
    DbvOctetString         *psOpaque;
  }                   uUpdtRecordId;
} DbvUpdtRecordId;

typedef struct DbvSuppliedRecord {
DbvUpdtRecordId            *psUpdtRecordId;
DbvUpdtSupplementalId      *psUpdtSupplementalId;
DbvCorrelationInfo         *psCorrelationInfo;
DbvASN1External             sRecord;
} DbvSuppliedRecord;

typedef struct DbvOriginPartToKeep {
DbvUpdtAction               eAction;
DbvInternationalString      sDatabaseName;
DbvObjectIdentifier         pcSchema;
DbvInternationalString     *psElementSetName;
} DbvOriginPartToKeep;

typedef struct DbvUpdateTaskPackage {
DbvOriginPartToKeep      sOriginPartToKeep;
DbvUpdateTargetPart      sTargetPart;
} DbvUpdateTaskPackage;
```

```
typedef struct DbvUpdateRequestPackage {
DbvOriginPartToKeep    sOriginPartToKeep;
DbvSuppliedRecord    **ppsSuppliedRecord;
} DbvUpdateRequestPackage;

typedef enum DbvUpdateReqOrRsp {
                    UROR_Request  = 1,
                    UROR_Response = 2
} DbvUpdateReqOrRsp;

typedef struct DbvUpdateExtendedService {
DbvUpdateReqOrRsp     eUpdateReqRsp;
union {
    DbvUpdateRequestPackage  *psUpdateRequest;
    DbvUpdateTaskPackage     *psUpdateResponse;
  }               uUpdateReqOrRsp;
} DbvUpdateExtendedService;

typedef struct DbvOriginData {
  DbvOriginEvent        eOriginEvent;
  union {
    DbvInitializeRsp           *psInitializeRsp;
    DbvSearchRsp               *psSearchRsp;
    DbvPresentRsp        *psPresentRsp;
    DbvDeleteResultSetRsp      *psDeleteResultSetRsp;
    DbvScanRsp                  *psScanRsp;
    DbvResourceReportRsp             *psResourceReportRsp;
    DbvSegmentReq         *psSegmentReq;
    DbvSortRsp                  *psSortRsp;
    DbvCloseReq                *psCloseReq;
    DbvExtendedServicesRsp  *psExtendedServicesRsp;
    DbvOctetString          *psInvalidAPDU;
  }   uOriginData;
} DbvOriginData;
```

```
typedef struct DbvTargetData {
  DbvTargetEvent  eTargetEvent;
  union {
    DbvInitializeReq          *psInitializeReq;
    DbvSearchReq              *psSearchReq;
    DbvPresentReq       *psPresentReq;
    DbvDeleteResultSetReq     *psDeleteResultSetReq;
    DbvScanReq                      *psScanReq;
    DbvResourceReportReq            *psResourceReportReq;
    DbvSortReq                      *psSortReq;
    DbvCloseReq                     *psCloseReq;
    DbvExtendedServicesReq    *psExtendedServicesReq;
    DbvReleaseReq       *peReleaseReq;
    DbvOctetString         *psInvalidAPDU;
  }               uTargetData;
} DbvTargetData;
```

# 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".<br>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.<br>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.<br>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().

# Referenzliste

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

[2] D-023 Test Tool Manual

[3] D-024 Installation and Operations